SQL

Structured Query Language (SQL ISO/CE

focused mainly on Oracle 11g r2 **for Data Science**

Vincent ISOZ

V6.0 r13
2018-12-09
oUUID 1839

Table Of Contents

1 Useful Links

The most important link as it gives you the possibility to use Oracle Enterprise online for free to train your skills:

http://www.oracle.com/technetwork/database/application-development/livesql/index.html

and finally, some other links:

http://www.google.com

http://www.youtube.com

http://sqlformat.org/

http://www.oracle.com/pls/db102/homepage (B1[1]-M1 Level)

http://sql.developpez.com (B1-M1 Level)

↳http://blog.developpez.com/sqlpro (B1-M1 Level)

http://www.oracle.com/technetwork/documentation/index.html#database

http://www.dba-ora.fr (B1 Level)

http://sql-plsql.blogspot.ch (B1-B2 Level)

https://forums.oracle.com/welcome (B1-M2 Level)

http://www.dba-oracle.com

https://www.video2brain.com/fr/formation/sql-les-fondamentaux (B1 Level)

http://www.w3schools.com/sql/sql_quiz.asp (B1 Level)

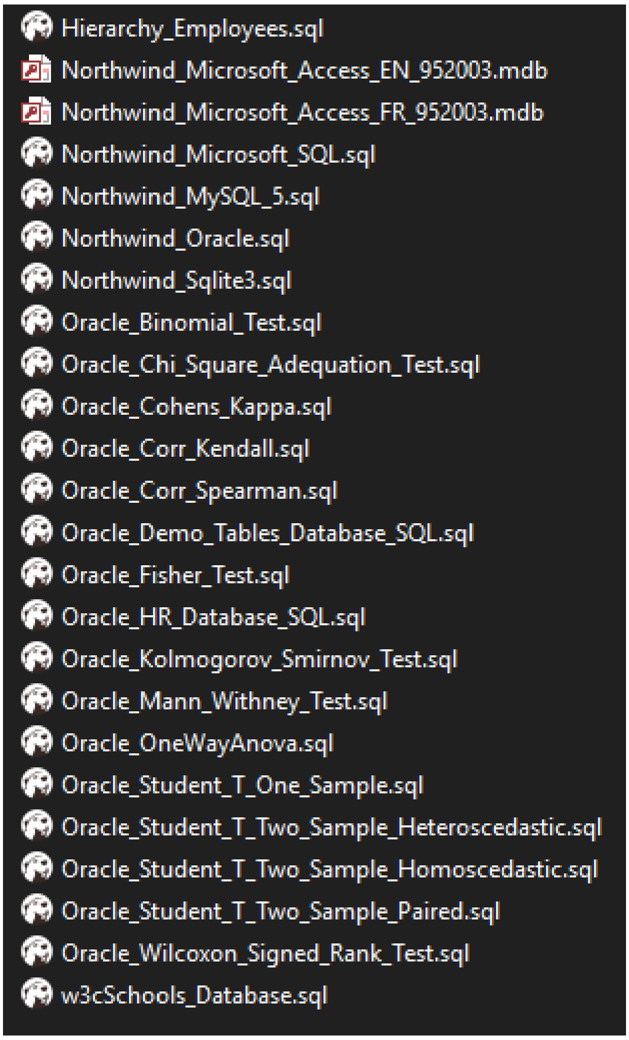
http://psoug.org (B1-B2 Level)

http://www.sqlines.com/online

[1] B1: first year Bachelor, B2: Second year of Bachelor, B3: Third year of Bachelor, M1: First year Master, M2: Second Year Master, Phd: " Philosophiæ doctor" level (=M2+[1;4])

2 Introduction

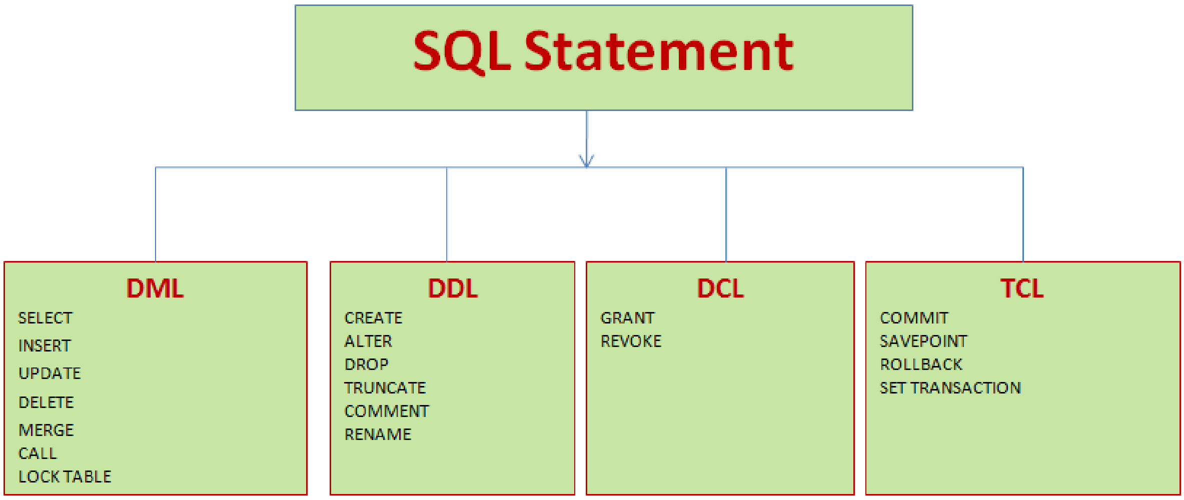
This PDF has for purpose to introduce the basics of SQL for Data Scientists in a 5 days training. The most important chapter for Data Scientists will be the last chapter at page 314. Database files are given only to the people that follow my courses.



SQL (Structured Query Language) is a special-purpose data query language designed for managing data held in a relational database management system (RDBMS). There are obviously (and sadly…) other data query languages, for example (for a more exhaustive list refer to Wikipedia):

- XPath
- DAX
- M
- Dplyr
- Data.table
- Panda
- JQuery
- …

Originally based upon relational algebra and tuple relational calculus, SQL consists mainly of a data definition language and a data manipulation language. The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control. Although SQL is often described as, and to a great extent is, a declarative language (4GL), it also includes procedural elements.



SQL was one of the first commercial languages for Edgar F. Codd's relational model, as described in his influential 1970 paper "A Relational Model of Data for Large Shared Data Banks". Despite not entirely adhering to the relational model as described by Codd, it became the most widely used database language

SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standards (ISO) in 1987. Since then, the standard has been enhanced several times with added features. Despite these standards, code is not completely portable among different database systems, which can lead to vendor lock-in. The different makers do not perfectly adhere to the standard, for instance by adding extensions, and the standard itself is sometimes ambiguous.

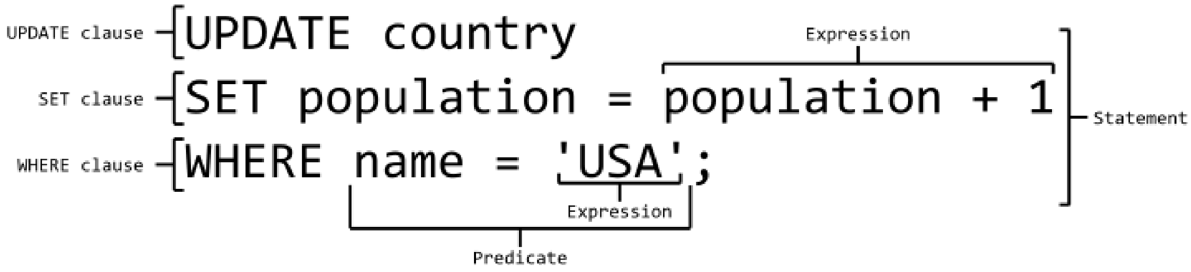
2.1 History

SQL was initially developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s when IBM created the first databases (on the bases of a paper written by the mathematician Edgar Franck Codd). This version, initially called SEQUEL (Structured English Query Language), was designed to manipulate and retrieve data stored in IBM's original quasi-relational database management system, System R, which a group at IBM San Jose Research Laboratory had developed during the 1970s. The acronym SEQUEL was later changed to SQL because "SEQUEL" was a trademark of the UK-based Hawker Siddeley aircraft company.

In the late 1970s, Relational Software, Inc. (now **Oracle Corporation**) saw the potential of the concepts described by Codd, Chamberlin, and Boyce and developed their own SQL-based RDBMS with aspirations of selling it to the U.S. Navy, Central Intelligence Agency, and other U.S. government agencies. In June 1979, Relational Software, Inc. introduced the first commercially available implementation of SQL, Oracle V2 (Version2) for VAX computers.

After testing SQL at customer test sites to determine the usefulness and practicality of the system, IBM began developing commercial products based on their System R prototype including System/38, SQL/DS, and DB2, which were commercially available in 1979, 1981, and 1983, respectively.

2.2 Syntax



The SQL language is subdivided into several language elements, including:

- Clauses, which are constituent components of statements and queries. (In some cases, these are optional.)
- Expressions, which can produce either scalar values, or tables consisting of columns and rows of data.
- Predicates, which specify conditions that can be evaluated to SQL three-valued logic (3VL) (true/false/unknown) orBoolean truth values and which are used to limit the effects of statements and queries, or to change program flow.
- Queries, which retrieve the data based on specific criteria. This is an important element of SQL.
- Statements, which may have a persistent effect on schemata and data, or which may control transactions, program flow, connections, sessions, or diagnostics.
- SQL statements also include the semicolon (";") statement terminator. Though not required on every platform, it is defined as a standard part of the SQL grammar.
- Insignificant whitespace is generally ignored in SQL statements and queries, making it easier to format SQL code for readability

2.3 Procedural extensions

SQL is designed for a specific purpose: to query data contained in a relational database. SQL is a set-based, declarative query language, not an imperative language like C or BASIC. However, there are extensions to Standard SQL which add procedural programming language functionality, such as control-of-flow constructs. These include:

Source	Common name	Full name
ANSI/ISO Standard	SQL/PSM	SQL/Persistent Stored Modules
Interbase / Firebird	PSQL	Procedural SQL
IBM DB2	SQL PL	SQL Procedural Language (implements SQL/PSM)
IBM Informix	SPL	Stored Procedural Language
Microsoft / Sybase	T-SQL	Transact-SQL
Mimer SQL	SQL/PSM	SQL/Persistent Stored Module (implements SQL/PSM)
MySQL	SQL/PSM	SQL/Persistent Stored Module (implements SQL/PSM)
Oracle	PL/SQL	Procedural Language/SQL (based on Ada)
PostgreSQL	PL/pgSQL	Procedural Language/PostgreSQL (based on Oracle PL/SQL)
PostgreSQL	PL/PSM	Procedural Language/Persistent Stored Modules (implements SQL/PSM)
Sybase	Watcom-SQL	SQL Anywhere Watcom-SQL Dialect
Teradata	SPL	Stored Procedural Language

Tableau 1Common Databases Technologies

In addition to the standard SQL/PSM extensions and proprietary SQL extensions, procedural and object-oriented programmability is available on many SQL platforms via DBMS integration with other languages. The SQL standard defines SQL/JRT extensions (SQL Routines and Types for the Java Programming Language) to support Java code in SQL databases. SQL Server 2005 uses the SQLCLR (SQL Server Common Language Runtime) to host managed .NET assemblies in the database, while prior versions of SQL Server were restricted to using unmanaged extended stored procedures that were primarily written in C. PostgreSQL allows functions to be written in a wide variety of languages including Perl, Python, Tcl, and C

2.4 Standardization

SQL was adopted as a standard by the American National Standards Institute (ANSI) in 1986 as SQL-86 and the International Organization for Standardization (ISO) in 1987. Nowadays the standard is subject to continuous improvement by the Joint Technical Committee ISO/IEC JTC 1, Information technology, Subcommittee SC 32, Data management and interchange which affiliate to ISO as well as IEC. It is commonly denoted by the pattern: ISO/IEC 9075-n:yyyy Part n: title, or, as a shortcut, ISO/IEC 9075.

ISO/IEC 9075 is complemented by ISO/IEC 13249: SQL Multimedia and Application Packages (SQL/MM) which defines SQL based interfaces and packages to widely spread applications like video, audio and spatial data.

Until 1996, the National Institute of Standards and Technology (NIST) data management standards program certified SQL DBMS compliance with the SQL standard. Vendors now self-certify the compliance of their products.

The original SQL standard declared that the official pronunciation for SQL is "es queue el". Many English-speaking database professionals still use the original pronunciation /ˈsiːkwəl/ (like the word "sequel"), including Donald Chamberlin himself.

The SQL standard has gone through a number of revisions:

Year	Name	Alias	Comments
1986	SQL-86	SQL-87	First formalized by ANSI.
1989	SQL-89	FIPS127-1	Minor revision, in which the major addition were integrity constraints. Adopted as FIPS 127-1.
1992	SQL-92	SQL2, FIPS 127-2	Major revision (ISO 9075), *Entry Level* SQL-92 adopted as FIPS 127-2.
1999	SQL:1999	SQL3	Added regular expression matching, recursive queries (e.g. transitive closure), triggers, support for procedural and control-of-flow statements, non-scalar types, and some object-oriented features (e.g. structured types). Support for embedding SQL in Java (SQL/OLB) and vice-versa (SQL/JRT).
2003	SQL:2003	SQL 2003	Introduced XML-related features (SQL/XML), *window functions*, standardized sequences, and columns with auto-generated values (including identity-columns).
2006	SQL:2006	SQL 2006	ISO/IEC 9075-14:2006 defines ways in which SQL can be used in conjunction with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database and publishing both XML and conventional SQL-data in XML form. In addition, it enables applications to integrate into their SQL code the use of XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.[37]
2008	SQL:2008	SQL 2008	Legalizes ORDER BY outside cursor definitions. Adds INSTEAD OF triggers. Adds the TRUNCATE statement.[38]
2011	SQL:2011		

Tableau 2 SQL Standard Evolution

2.5 Well Know RDBMS using SQL

- 4e Dimension (4D)
- Microsoft Access
- Adonix X3
- OpenOffice Base
- DB2 (AS400)
- Firebird
- Visual FoxPro
- HyperFileSQL
- Informix
- Ingres
- MariaDB
- MaxDB (anciennement SAP db)
- Microsoft SQL Server
- Mimer
- MySQL
- Ocelot
- Oracle
- Paradox
- PostgreSQL
- SQLite
- SQL/MM
- Sybase
- Teradata
- Microsoft Excel
- HSQLDB
- CUBRID
- H2
- ...

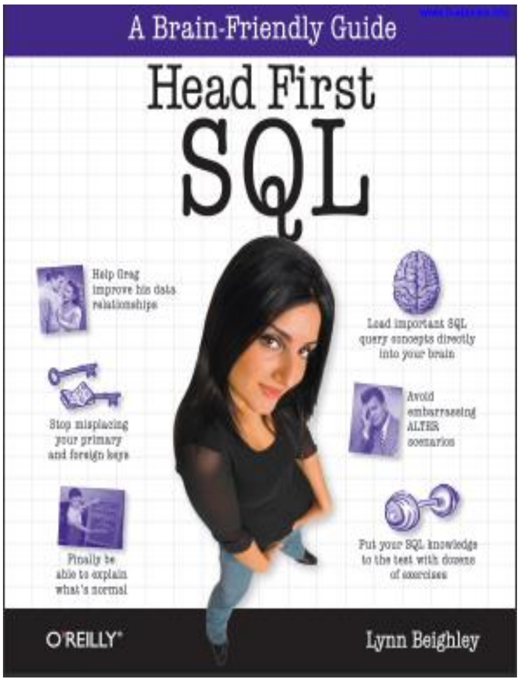
All these systems have some particularities which some are not found in others. Moreover, it is always interesting to refer to the reference manual RDBMS during special or complex queries, as well as their optimization.

2.6 Why IBM Oracle at University?

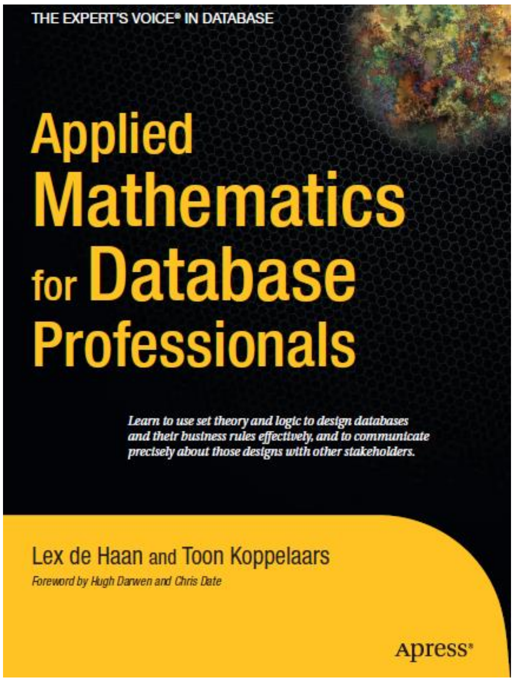
- The computer scientists that created the basics of database theory were working for IBM
- Oracle has indexes choices that are much more interesting for advanced data management
- The SQL language of Oracle has graduate statistical functions that others don't have
- In general, it is accepted that Oracle is more robust than others systems

2.7 Recommended References

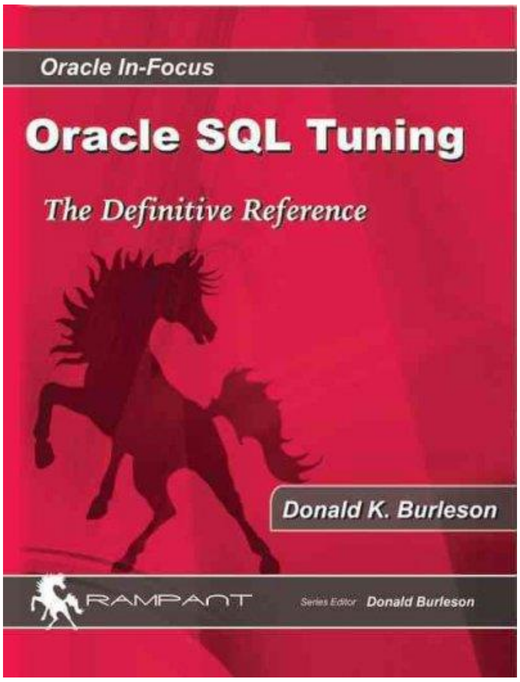
Excepted the ISO reference book I strongly recommend the further lectures:



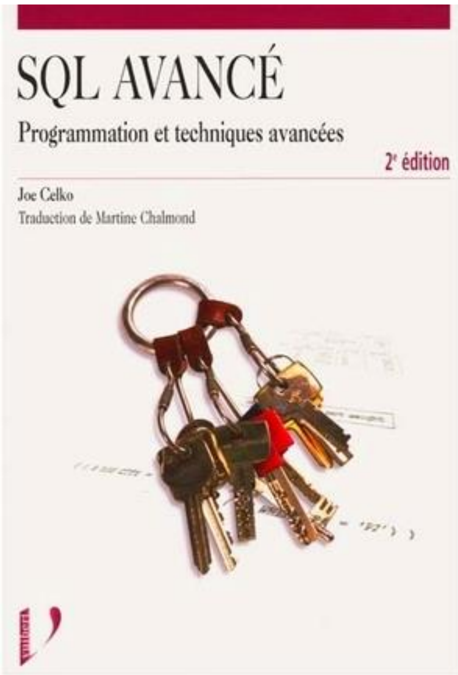
B1 Level



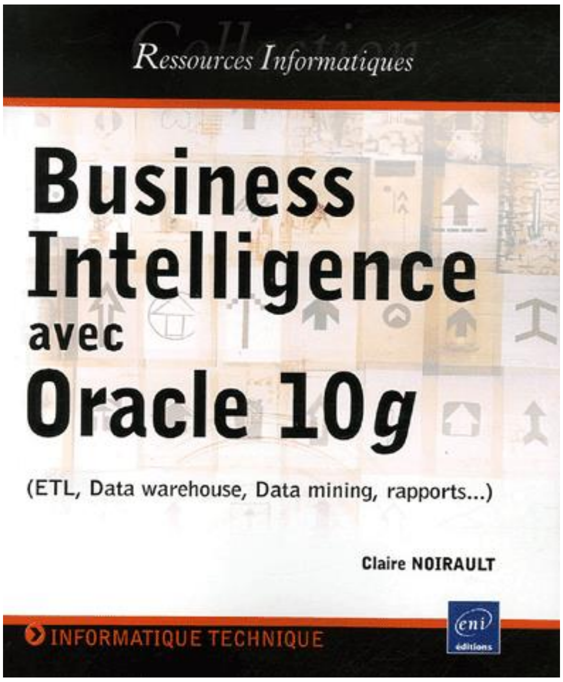
B1-B2 Level



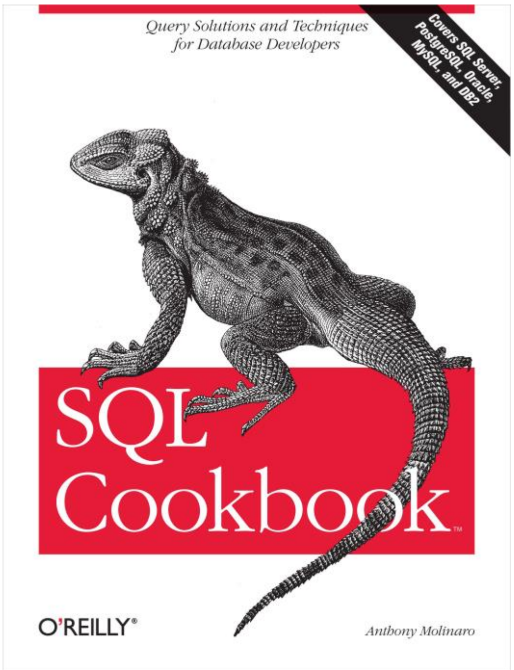
B1-B3 Level



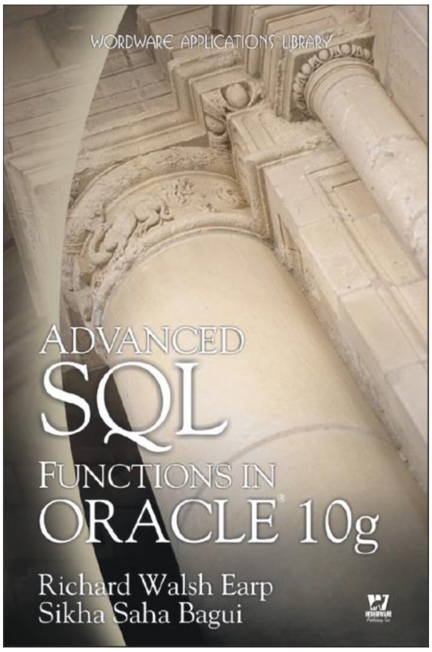
B1-M1 Level



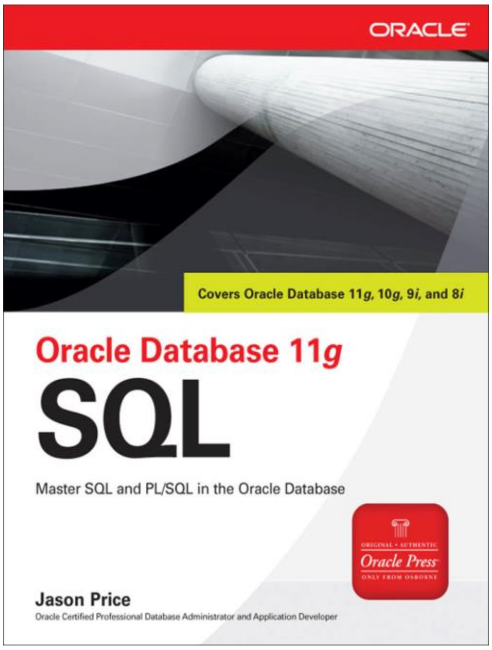
B1-PhD Level



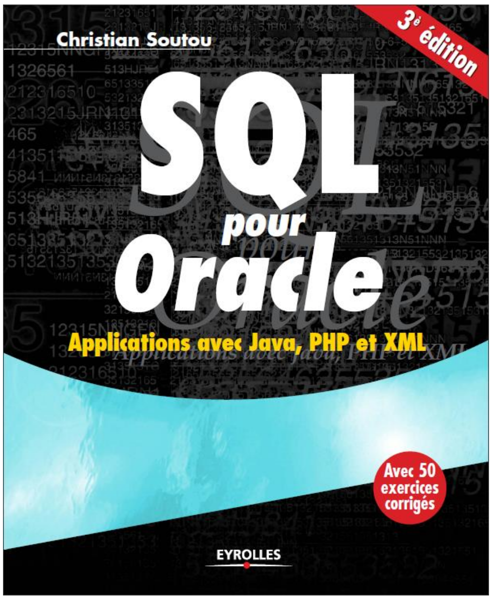
B1-B2 Level



B1-PhD Level



B1-PhD Level


Christian Soutou
3e édition
SQL
pour
Oracle
Applications avec Java, PHP et XML
Avec 50 exercices corrigés
EYROLLES

B1-B3 Level

3 Lenszynski-Reddick Naming convention

As said during the MS Office Access training if you do not follow the LR naming convention for your objects you will have troubles to read large SQL queries. In the present e-book I did not follow this convention to prove you the problem of lecture that the non-respect of this convention can imply:

http://en.wikipedia.org/wiki/Leszynski_naming_convention

4 SQL for DML (Data Manipulation Language)

Go on:

http://www.w3schools.com/sql/

to use the on-line simple SQL query.

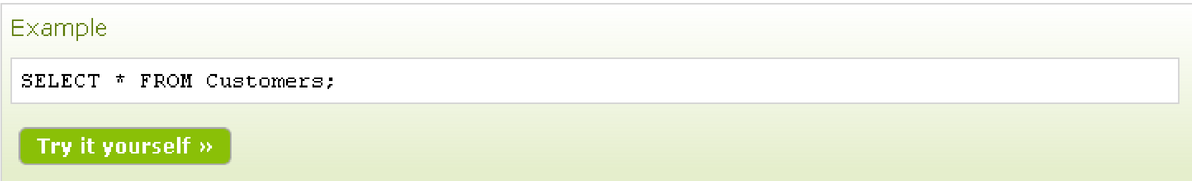
SQL is a standard language for accessing databases.

Our SQL tutorial will teach you how to use SQL to access and manipulate data in: MySQL, SQL Server, Access, Oracle, Sybase, DB2, and other database systems.

With our online SQL editor, you can edit the SQL statements, and click on a button to view the result.

Example:

```
SELECT * FROM Customers;
```



Click on the "Try it yourself" button to see how it works.

In this tutorial we will use the well-known Northwind sample database (included in MS Access and MS SQL Server).

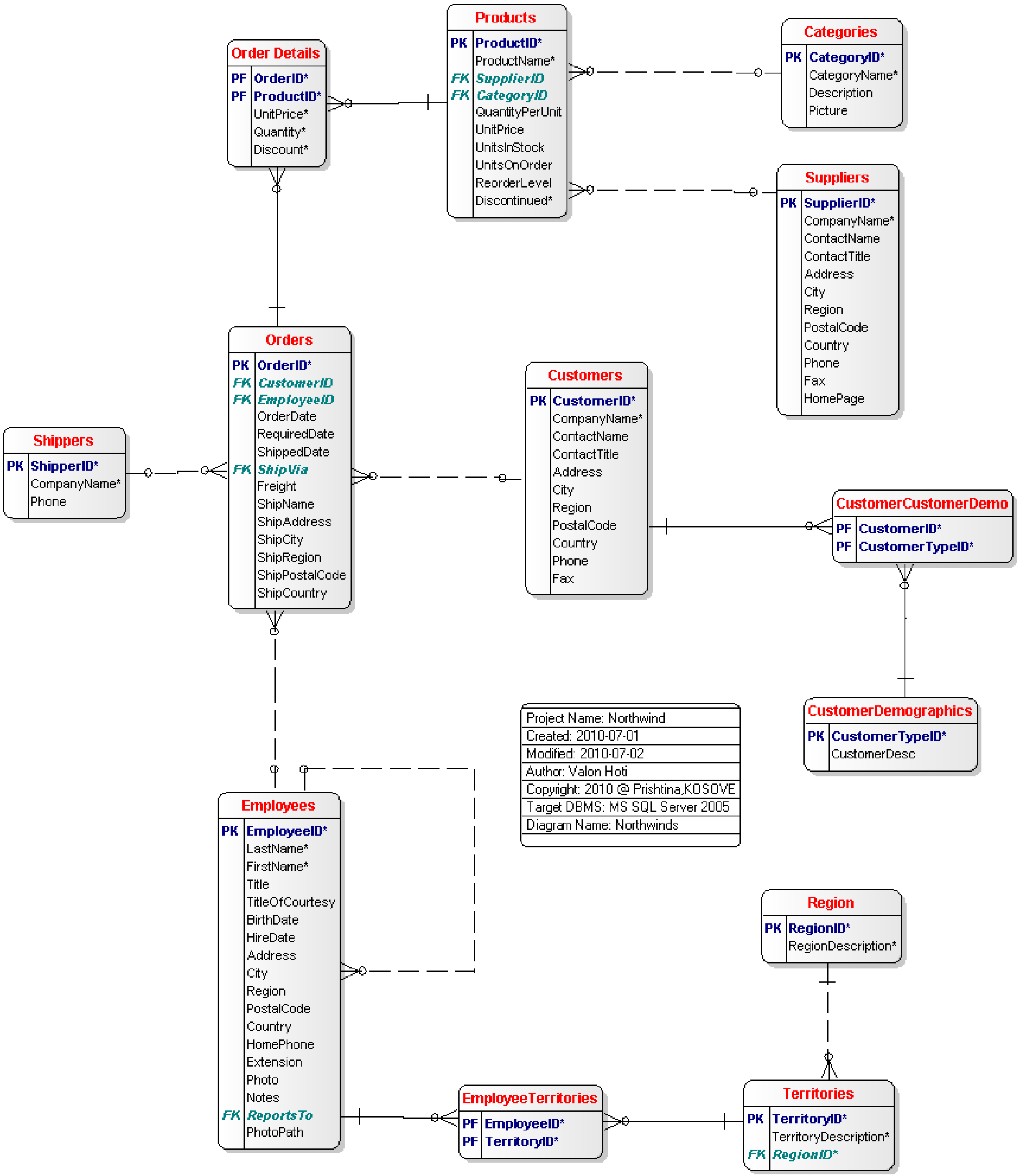


Figure 1 Northwind Database "star schema"

Keep in Mind That... SQL is NOT case sensitive: "SELECT" is the same as "select"

Semicolon after SQL Statements?

Some database systems require a semicolon at the end of each SQL statement. Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server. In this tutorial, we will use semicolon at the end of each SQL statement.

4.1 Comments IN SQL

In Oracle, comments may be introduced in two ways:

With /*...*/, as in C.

With a line that begins with two dashes --.

Thus:

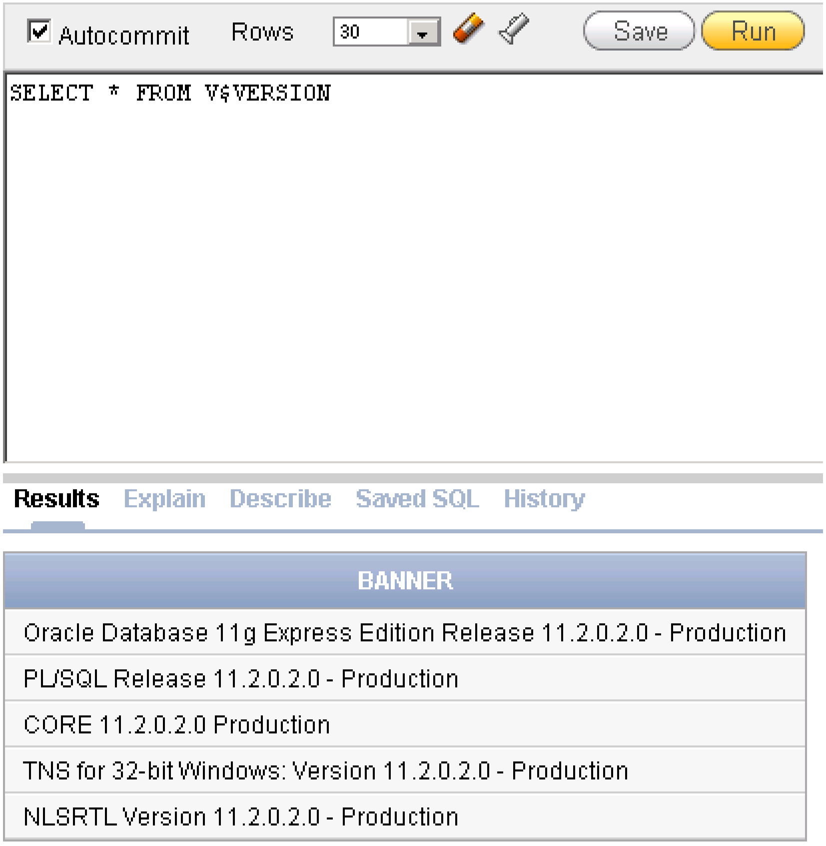
```
-- This is a comment
SELECT * /* and so is this */
FROM R;
```

4.2 SQL Version

To see the actual version of you SQL Engine (because depending on the version some functions/statements won't work), in MySQL use:

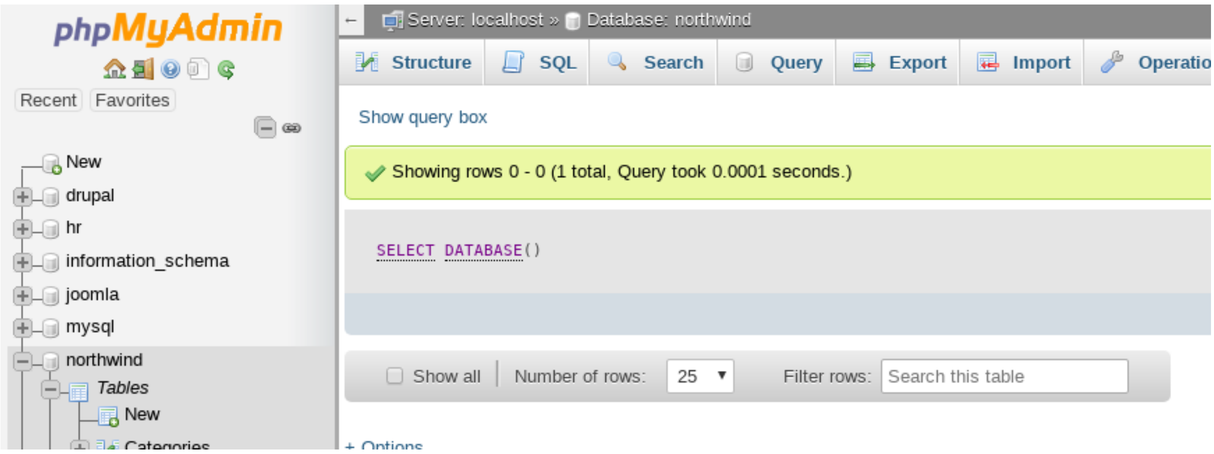
```
SELECT @@version;
```

and in Oracle:

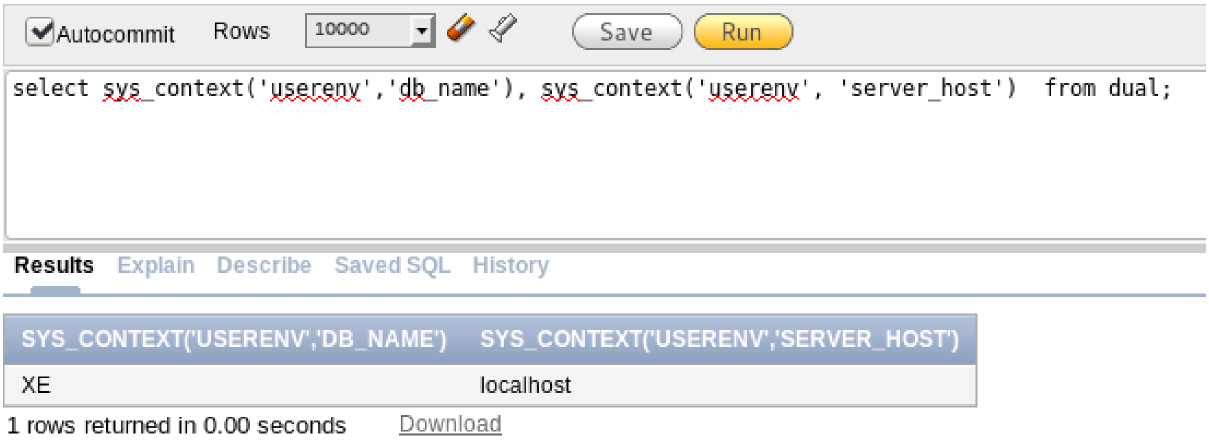


4.3 SQL SELECT Statement

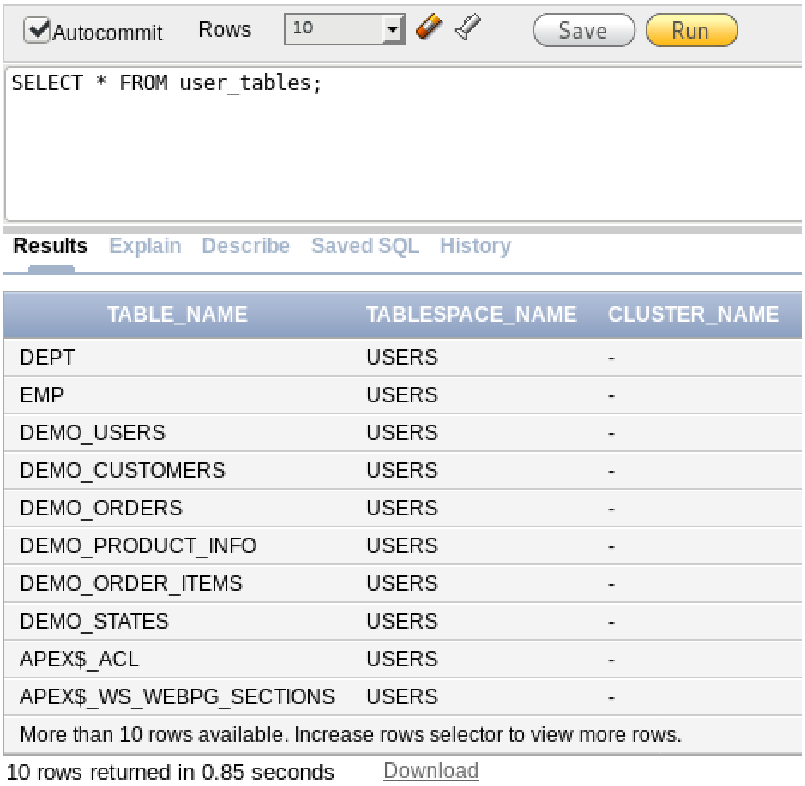
The SELECT statement is used mainly to select data from a database. But it can also be used to get information on the active database. For example, on MySQL:



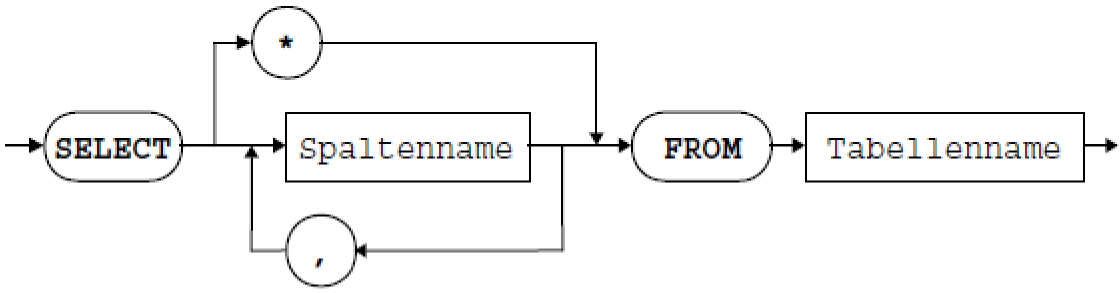
And on Oracle:



And still on oracle to get all tables of the actual database use:



The result is stored in a result table, called the result-set.



SQL SELECT Syntax:

```
SELECT column_name,column_name
FROM table_name;
```

and:

```
SELECT * FROM table_name;
```

When you select only a few columns, we say we're using an "SQL projection"...

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany

2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

The following SQL statement selects the "CustomerName" and "City" columns from the "Customers" table:

```
SELECT CustomerName,City FROM Customers;
```

The following SQL statement selects all the columns from the "Customers" table:

```
SELECT * FROM Customers;
```

4.4 SQL USE Statement

If you have multiple databases on your server you could have to specify which database you want to use (MySQL):

```
USE dbNorthwind

SELECT CustomerName,City FROM Customers;
```

Or depending on the technology here you can see an example of the beginning of a query using two tables of two different tables (SQL Server):

```
SELECT * FROM Accounts.dbo.TableOfAccounts ,Sales.dbo.TableOfSales....
```

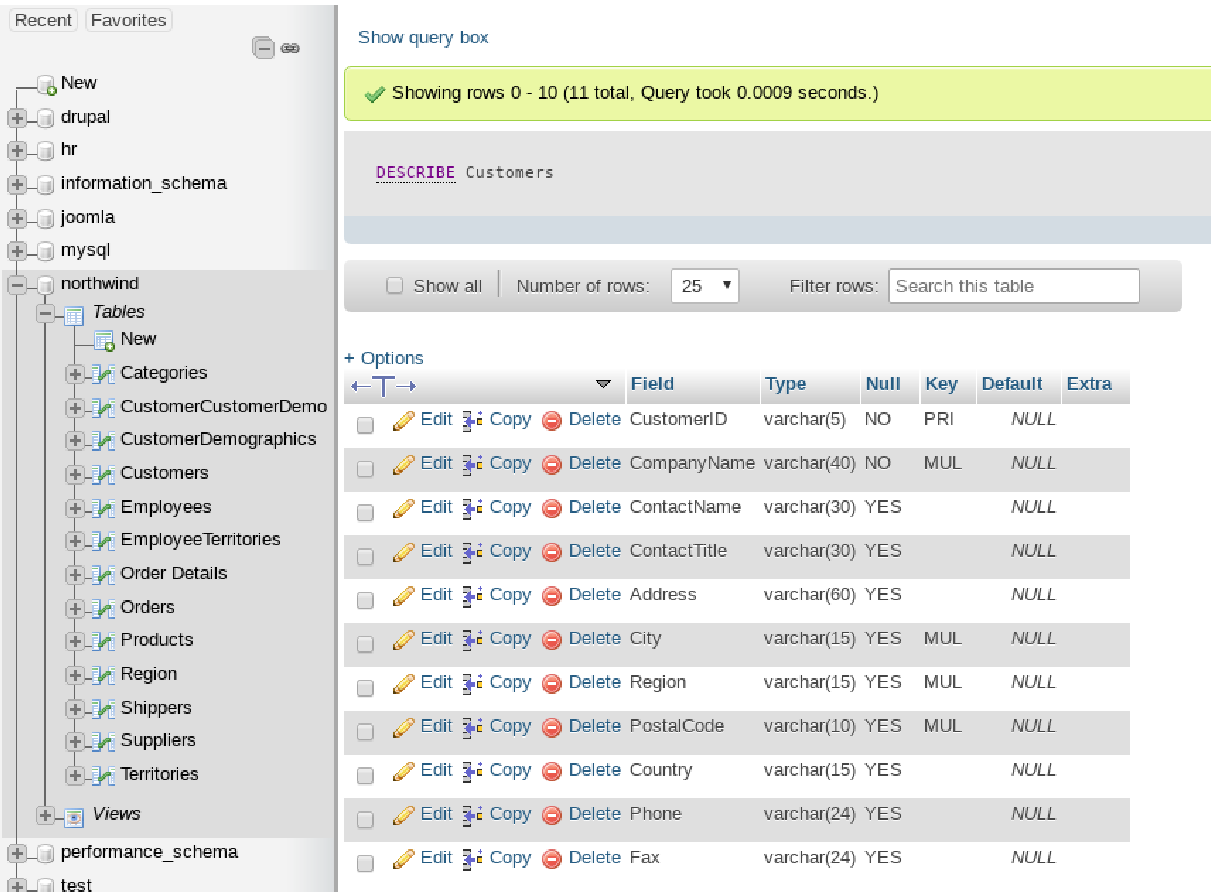
With Oracle you have to change the user scheme using:

```
ALTER SESSION SET CURRENT_SCHEMA=other user
```

where *other user* is the name of an another (**without quotes!**) user who has access to another scheme.

4.4.1 SQL DESCRIBE

To get the technical details about a table, on mySQL use the statement DESCRIBE:



Identically for Oracle:

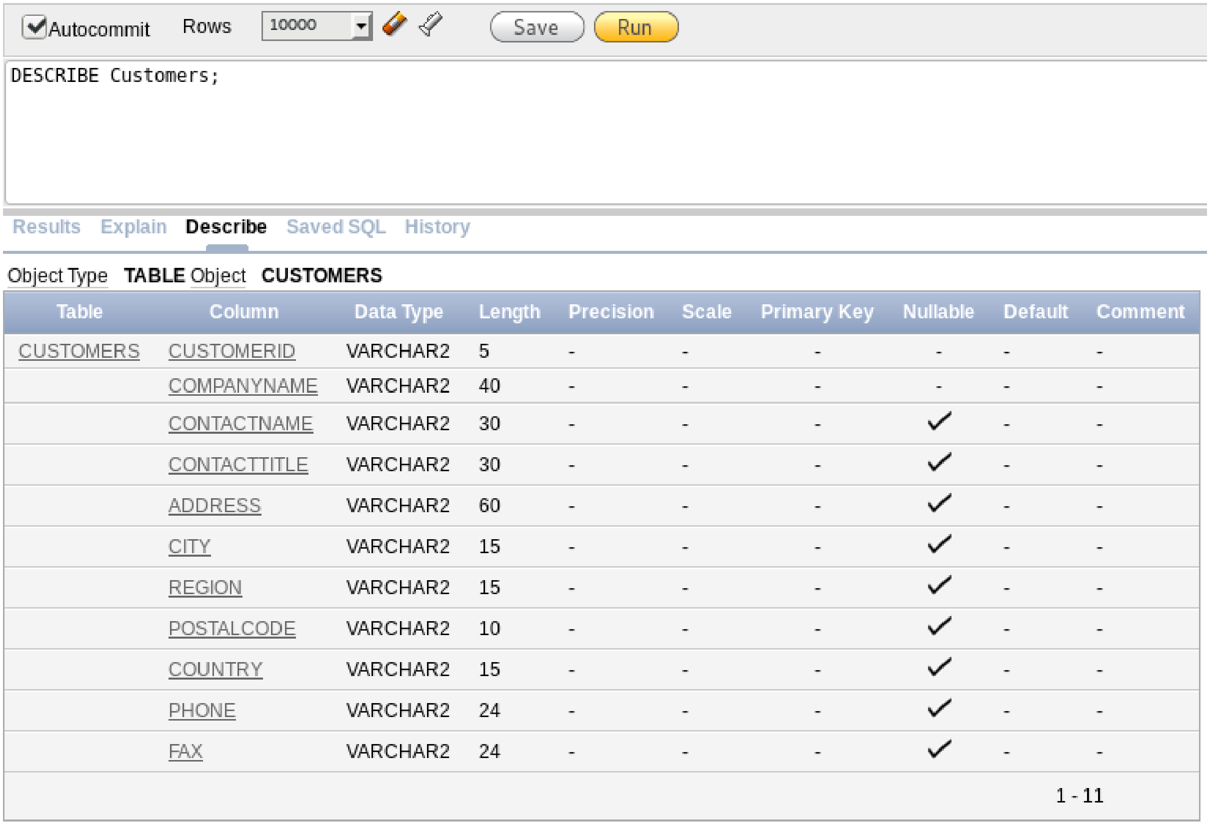


Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
CUSTOMERS	CUSTOMERID	VARCHAR2	5	-	-	-	-	-	-
	COMPANYNAME	VARCHAR2	40	-	-	-	-	-	-
	CONTACTNAME	VARCHAR2	30	-	-	-	✓	-	-
	CONTACTTITLE	VARCHAR2	30	-	-	-	✓	-	-
	ADDRESS	VARCHAR2	60	-	-	-	✓	-	-
	CITY	VARCHAR2	15	-	-	-	✓	-	-
	REGION	VARCHAR2	15	-	-	-	✓	-	-
	POSTALCODE	VARCHAR2	10	-	-	-	✓	-	-
	COUNTRY	VARCHAR2	15	-	-	-	✓	-	-
	PHONE	VARCHAR2	24	-	-	-	✓	-	-
	FAX	VARCHAR2	24	-	-	-	✓	-	-

1 - 11

4.4.2 SQL Aliases

SQL aliases are used to give a database table, or a column in a table, a temporary name.

Basically, aliases are created to make column names more readable.

SQL Alias Syntax for Columns:

```
SELECT column_name AS alias_name
FROM table_name;
```

SQL Alias Syntax for Tables (very useful for joins):

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico

3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
....						

And a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10643	1	6	1997-08-25	1
10644	88	3	1997-08-25	2
10645	34	4	1997-08-26	1
...				

The following SQL statement specifies two aliases, one for the CustomerName column and one for the ContactName column.

Tip: It require double quotation marks or square brackets if the column name contains spaces:

```
SELECT CustomerName AS Customer, ContactName AS [Contact Person]
FROM Customers;
```

It will give:

Customer	Contact Person
Alfreds Futterkiste	Maria Anders
Ana Trujillo Emparedados y helados	Ana Trujillo
Antonio Moreno Taquería	Antonio Moreno
Around the Horn	Thomas Hardy
Berglunds snabbköp	Christina Berglund
Blauer See Delikatessen	Hanna Moos
...	

In the following SQL statement, we combine four columns (Address, City, PostalCode, and Country) and create an alias named "Address":

```
SELECT CustomerName, Address+', '+City+', '+PostalCode+', '+Country AS
Address
FROM Customers;
```

it will give:

CustomerName	Address
Alfreds Futterkiste	Obere Str. 57, Berlin, 12209, Germany

Ana Trujillo Emparedados y helados	Avda. de la Constitución 2222, México D.F., 05021, Mexico
Antonio Moreno Taquería	Mataderos 2312, México D.F., 05023, Mexico
Around the Horn	120 Hanover Sq., London, WA1 1DP, UK
Berglunds snabbköp	Berguvsvägen 8, Luleå, S-958 22, Sweden
Blauer See Delikatessen	Forsterstr. 57, Mannheim, 68306, Germany
...	

The following SQL statement selects all the orders from the customer "Alfreds Futterkiste". We use the "Customers" and "Orders" tables, and give them the table aliases of "c" and "o" respectively (Here we have used aliases to make the SQL shorter):

```
SELECT o.OrderID, o.OrderDate
FROM Orders AS o;
```

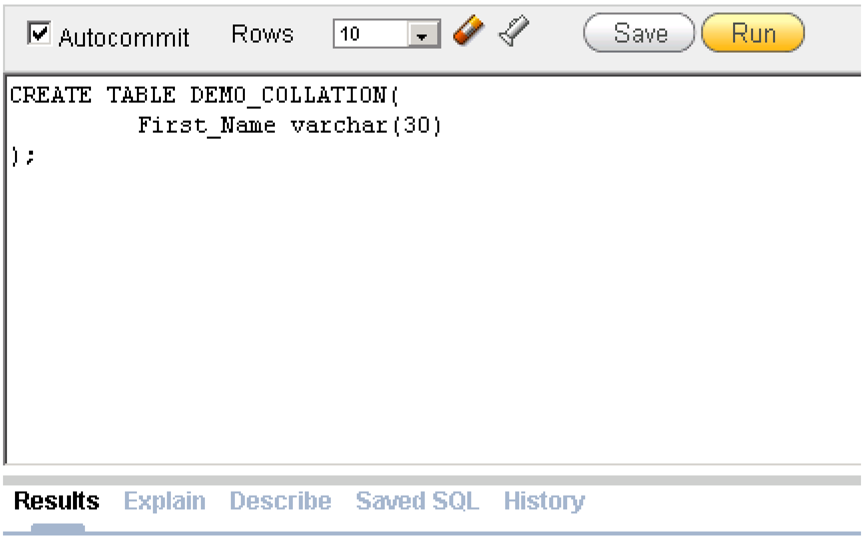
Aliases can also be useful when:

- There are more than one table involved in a query (see later JOINS)
- Functions are used in the query
- Column names are big or not very readable
- Two or more columns are combined together

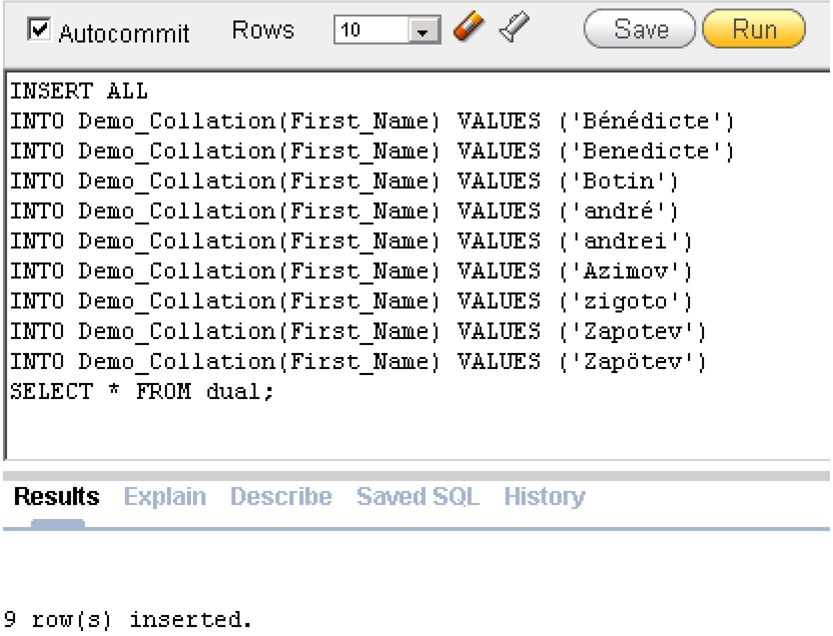
4.4.3 SQL COLLATION Statement

With the COLLATE clause, you can override whatever the default collation is for a comparison. COLLATE may be used in various parts of SQL statements.

To see what is collation we will focus on Oracle. First create the following table:



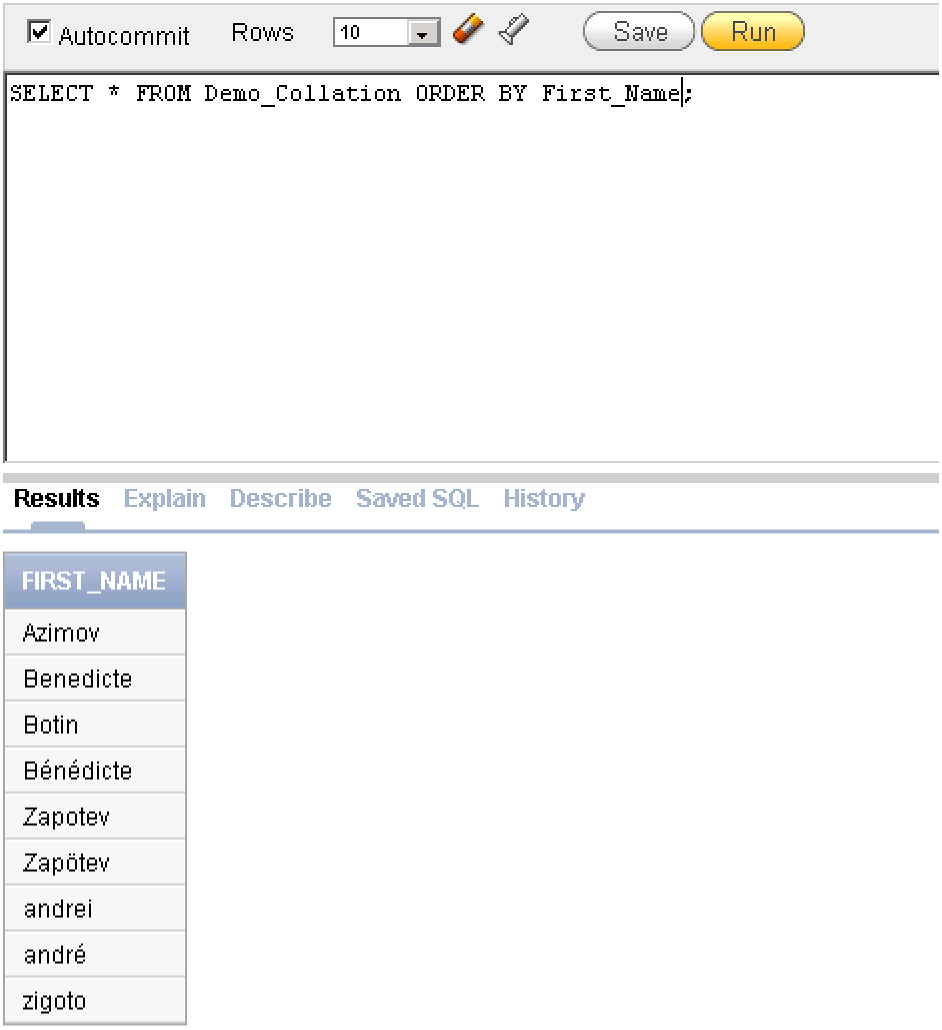
and add some values:




```sql
INSERT ALL
INTO Demo_Collation(First_Name) VALUES ('Bénédicte')
INTO Demo_Collation(First_Name) VALUES ('Benedicte')
INTO Demo_Collation(First_Name) VALUES ('Botin')
INTO Demo_Collation(First_Name) VALUES ('andré')
INTO Demo_Collation(First_Name) VALUES ('andrei')
INTO Demo_Collation(First_Name) VALUES ('Azimov')
INTO Demo_Collation(First_Name) VALUES ('zigoto')
INTO Demo_Collation(First_Name) VALUES ('Zapotev')
INTO Demo_Collation(First_Name) VALUES ('Zapötev')
SELECT * FROM dual;
```

```
9 row(s) inserted.
```

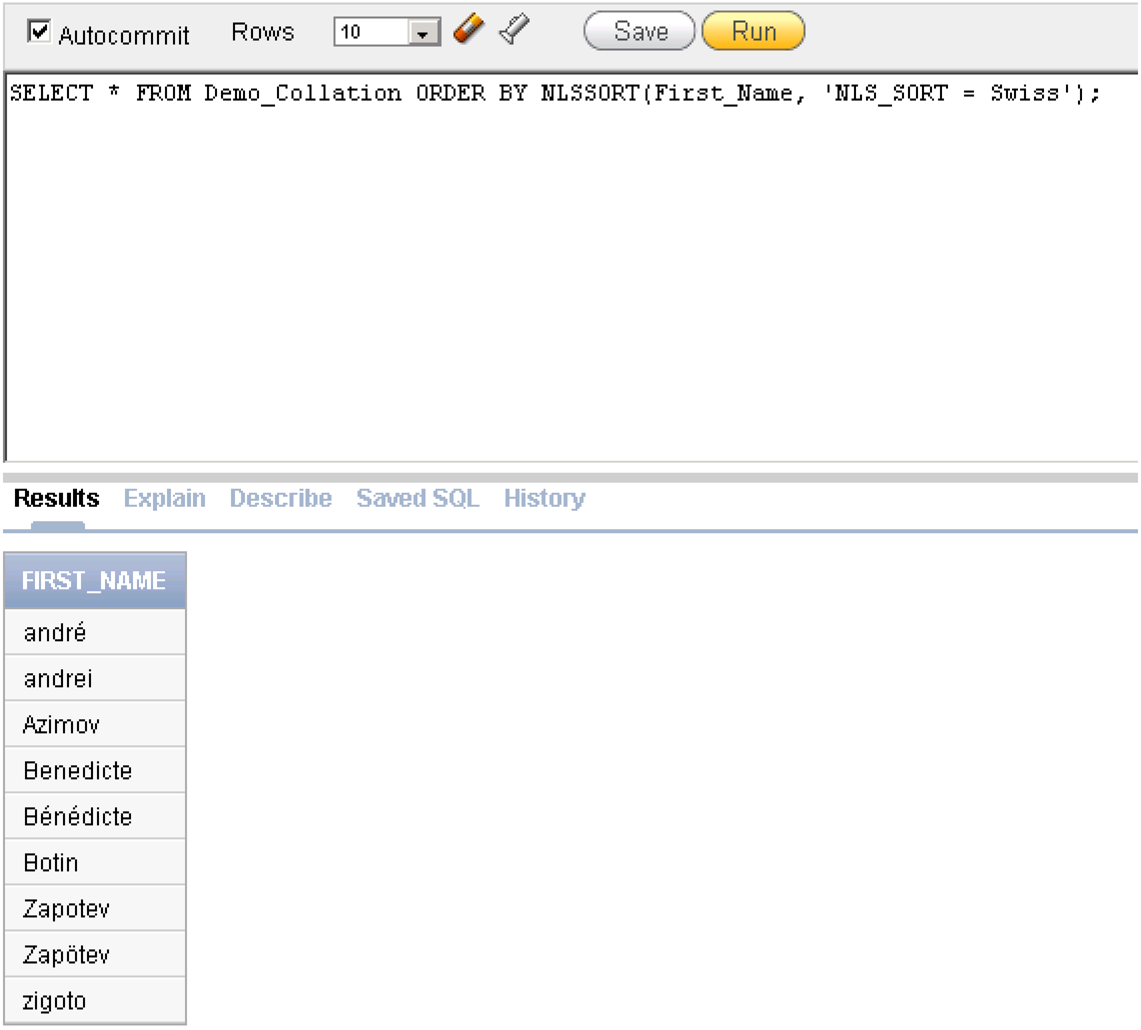
Now run the following query:



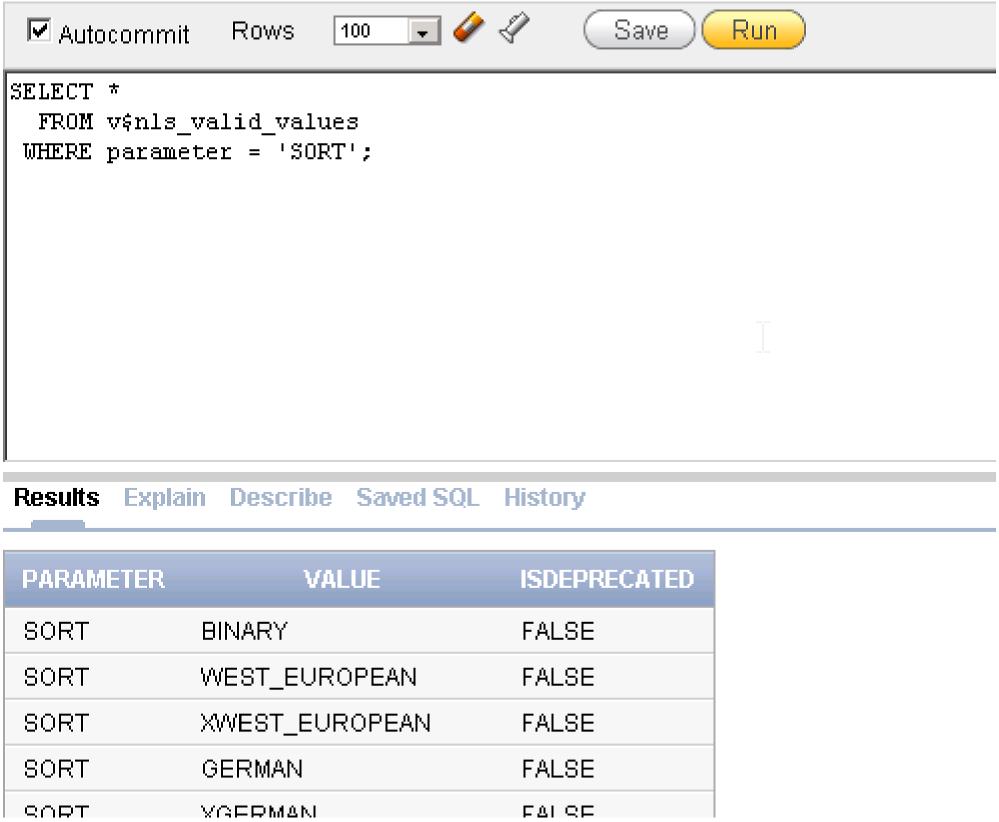

```sql
SELECT * FROM Demo_Collation ORDER BY First_Name;
```

FIRST_NAME
Azimov
Benedicte
Botin
Bénédicte
Zapotev
Zapötev
andrei
andré
zigoto

As you can see the order is return given the binary ASCII code of character. To have a more suitable result corresponding to your language you have to specify your collation using **National Language Support (NLS)** statement:



You can see all available collation by running the following query:



In SQL Server the syntax is almost very different!

4.4.4 SQL random sample

When you do Data Mining (supervised learning machine) you have to select a sample of your tables.

Here is the syntax to take a random sample of 1000 rows on SQL Server:

```
SELECT * FROM Sales.SalesOrderDetail TABLESAMPLE (1000 ROWS)
```

and the same on **Oracle**:

```
SELECT   *
FROM     (
         SELECT  *
         FROM DEMO_ORDER_ITEMS
         ORDER BY
                 dbms_random.value
         )
WHERE rownum <= 10
```

Here is the syntax to take a random sample of 25% percent of the total number of rows in **Oracle**:

```
SELECT * FROM DEMO_ORDER_ITEMS SAMPLE(25)
```

4.5 SQL UNION

The UNION operator is used to combine the result-set of two or more SELECT statements.

Notice that each SELECT statement within the UNION must have the same number of columns. The columns must also have similar data types. Also, the columns in each SELECT statement must be in the same order.

SQL UNION Syntax:

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

Note: The UNION operator selects only distinct values by default. To allow duplicate values, use the ALL keyword with UNION.

SQL UNION ALL Syntax:

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

Note: The column names in the result-set of a UNION are usually equal to the column names in the first SELECT statement in the UNION.

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

And a selection from the "Suppliers" table:

SupplierID	SupplierName	ContactName	Address	City	PostalCode	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	Londona	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA

The following SQL statement selects all the **different** cities (only distinct values) from the "Customers" and the "Suppliers" tables:

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

Note: UNION cannot be used to list ALL cities from the two tables. If several customers and suppliers share the same city, each city will only be listed once. UNION selects only distinct values. Use UNION ALL to also select duplicate values!

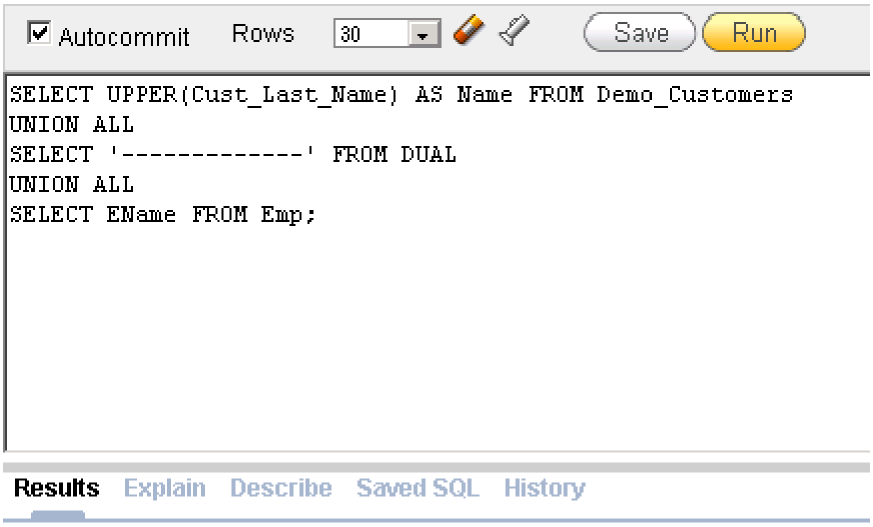
The following SQL statement uses UNION ALL to select **all** (duplicate values also) cities from the "Customers" and "Suppliers" tables:

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

The following SQL statement uses UNION ALL to select **all** (duplicate values also) **German** cities from the "Customers" and "Suppliers" tables:

```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION ALL
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

With Oracle you can do something sometimes interesting by adding à separation line that seems **works only with UNION ALL**:



NAME
BRADLEY
DULLES
HARTSFIELD
LAGUARDIA
LAMBERT
LOGAN
OHARE

KING
BLAKE
CLARK
JONES
SCOTT
FORD
SMITH

4.6 SQL SELECT DISTINCT and DISTINCTROWStatement

In a table, a column may contain many duplicate values; and sometimes you only want to list the different (distinct) values.

The DISTINCT keyword can be used to return only distinct (different) values.

SQL SELECT DISTINCT Syntax:

```
SELECT DISTINCT column_name,column_name
FROM table_name;
```

The following SQL statement selects only the distinct **values** from the "City" columns from the "Customers" table:

```
SELECT DISTINCT City FROM Customers;
```

The following SQL statement, that seems to exist only in Microsoft Access, selects only the distinct **records** from the **whole table** (including also not visible columns from the SELECT statement) the "City" columns from the "Customers" table:

```
SELECT DISTINCTROW City FROM Customers;
```

4.7 SQL WHERE Clause

The WHERE clause is used to extract only those records that fulfill a specified criterion.

SQL WHERE Syntax:

```
SELECT column_name,column_name
FROM table_name
WHERE column_name operator value;
```

The following SQL statement selects all the customers from the country "Mexico", in the "Customers" table:

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

SQL requires single quotes around text values (most database systems will also allow double quotes).

However, numeric fields should not be enclosed in quotes:

```
SELECT * FROM Customers
WHERE CustomerID=1;
```

The following operators can be used in the WHERE clause:

Operator	Description
=	Equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Tableau 3 Logical Operators

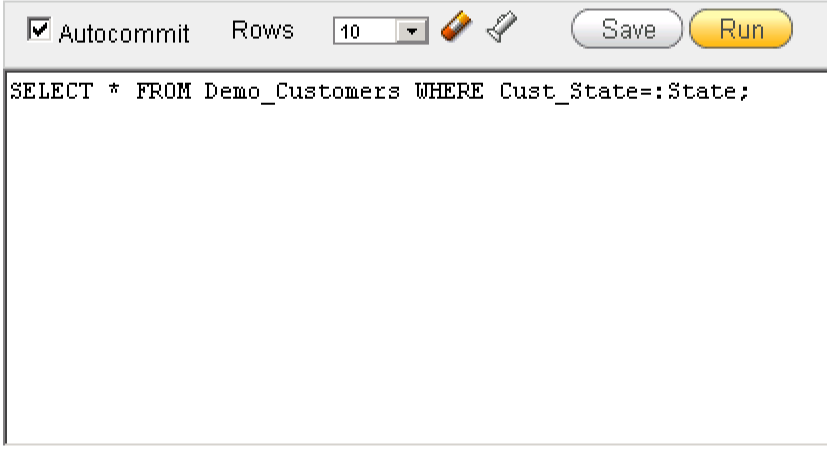
4.7.1 WHERE with interactive parameters

With Oracle you have the possibility to make the queries interactive. This is used a lot in financial and predictive models.

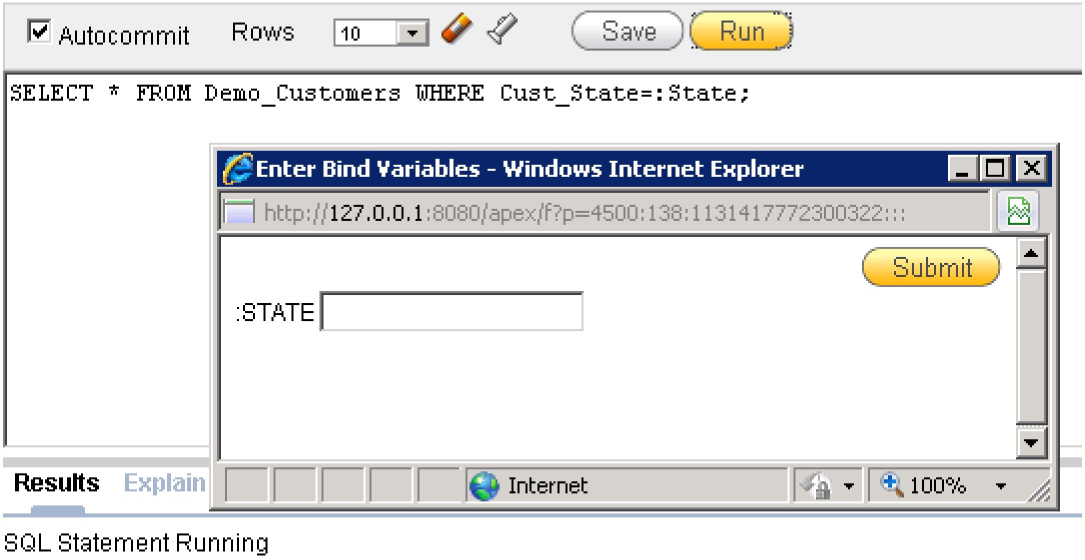
To do this just write in query constant criterias the following:

:OneWord

for example, as:

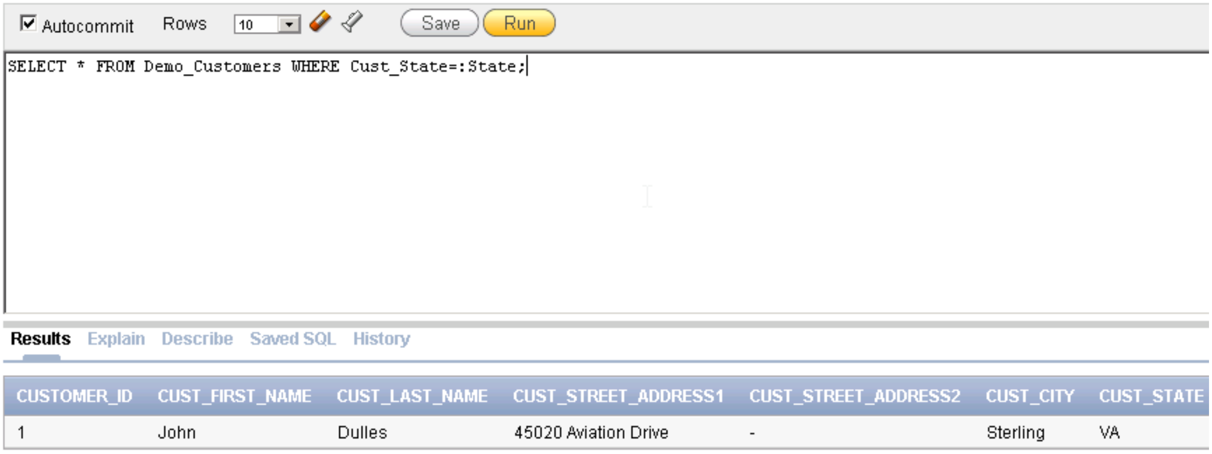


The when you click on **Run** you will get:



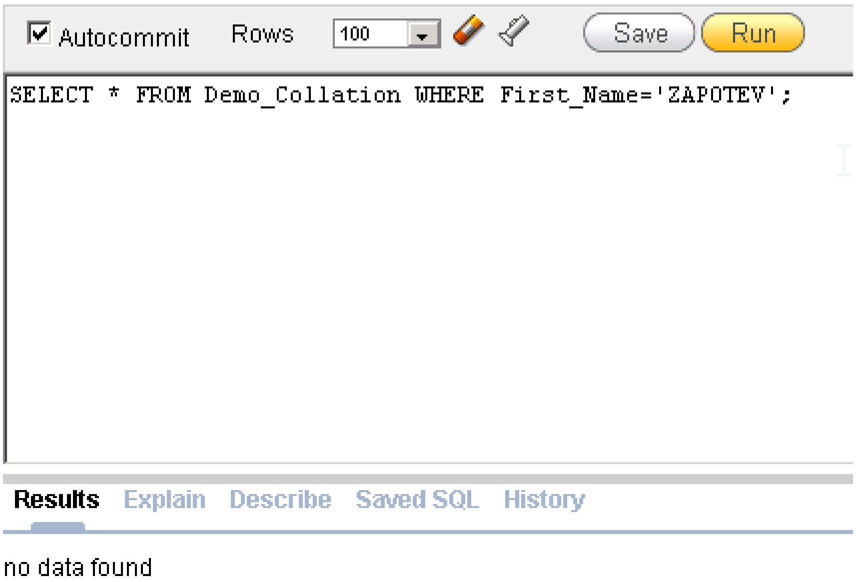
you type then a value:

and clic on **Submit** to get:

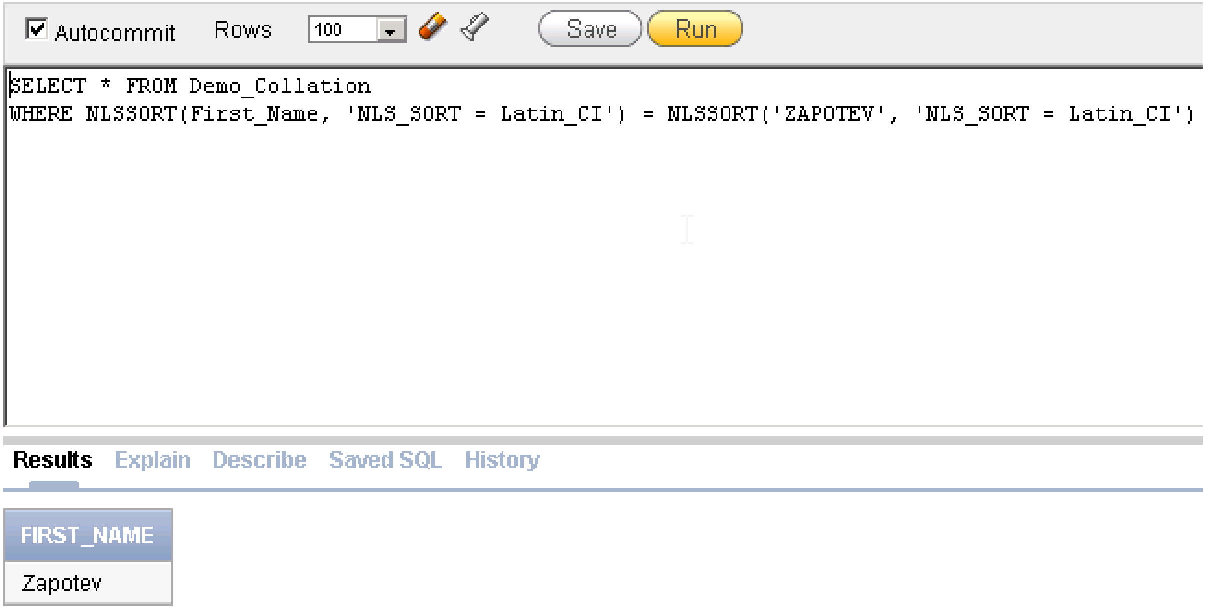


4.7.2 WHERE using COLLATION

In Oracle run now the following query based on the table created before:

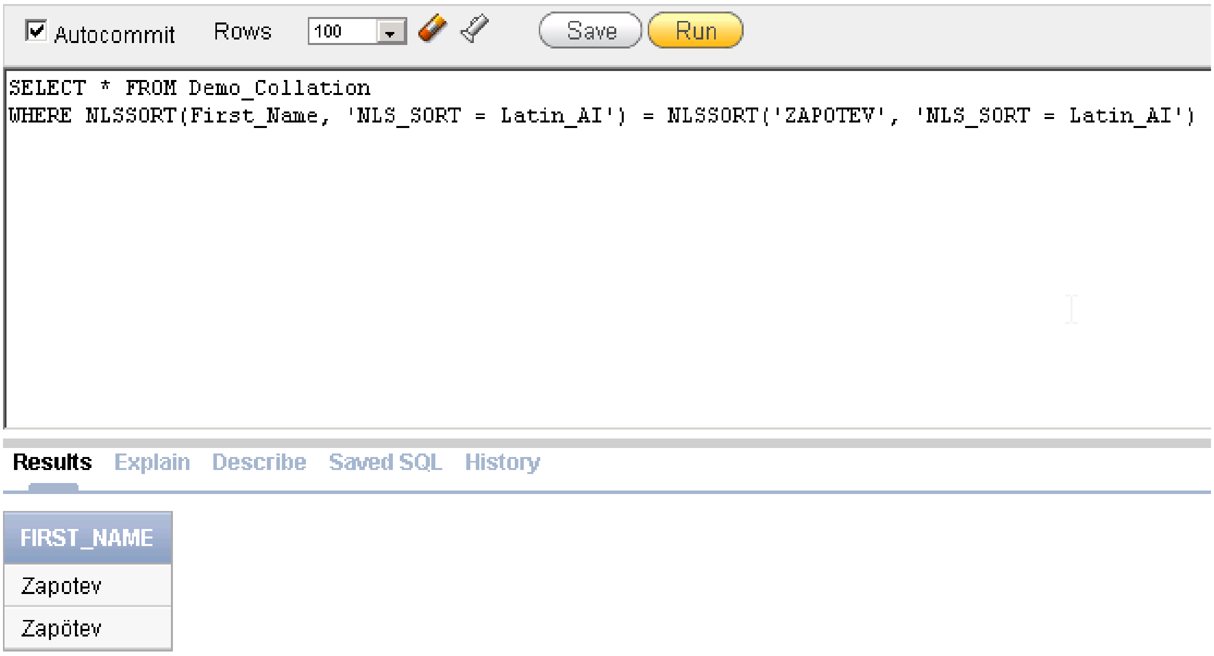


As you can see the system is case sensitive. Now type:



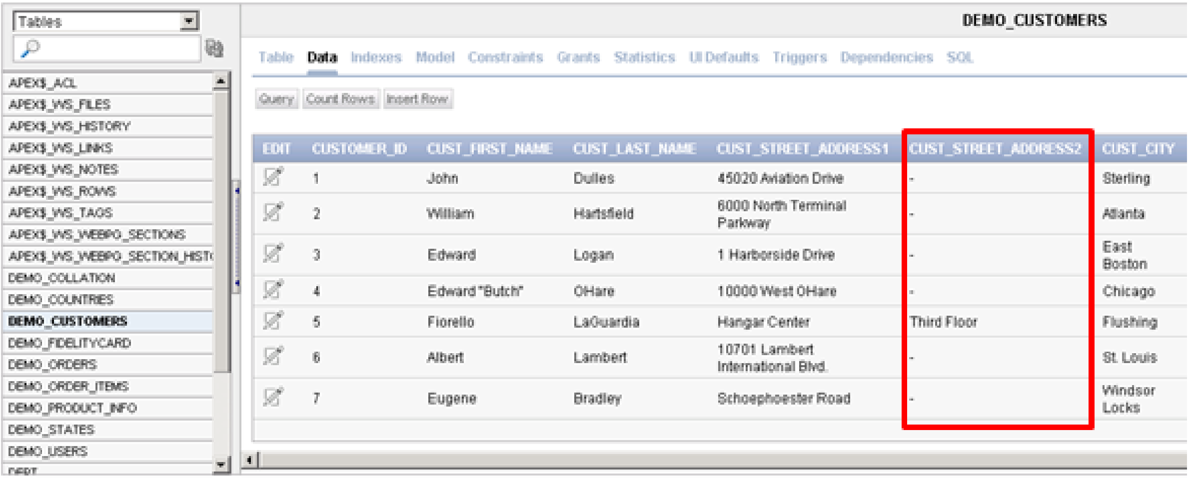
As you can see the system in now case insensitive (CI) but still sensitive to accents!

To make the query case insensitive and accent insensitive just write:

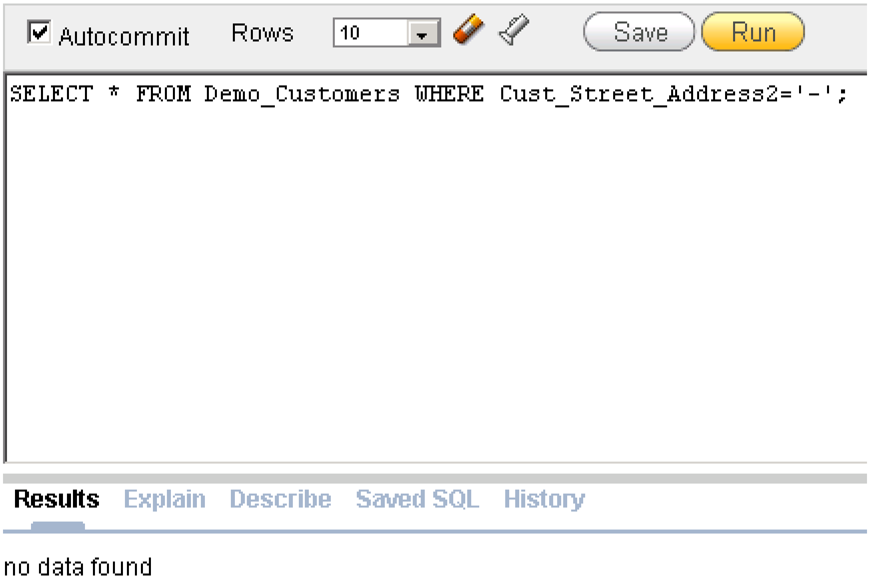


4.7.3 WHERE using IS NULL or IS NOT NULL

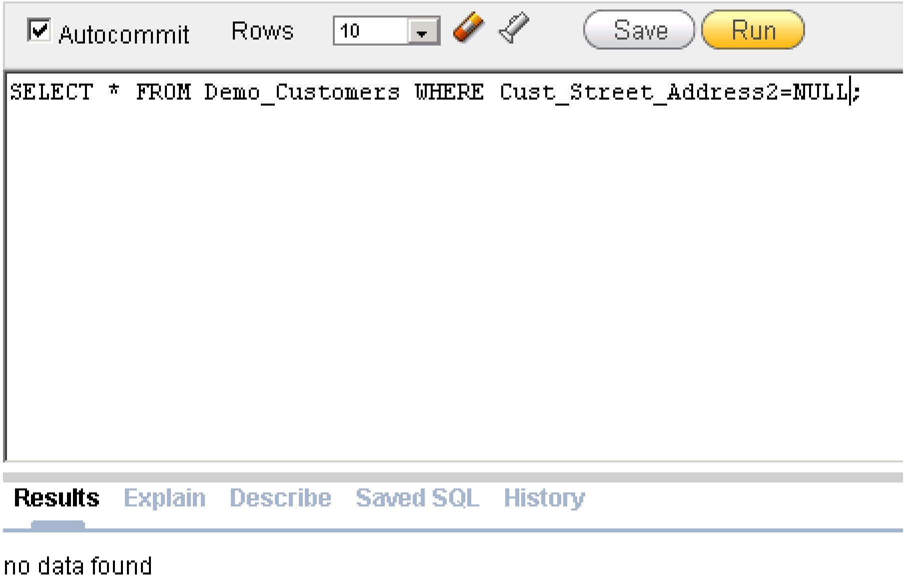
Tables like the following in Oracle have empty "cells":



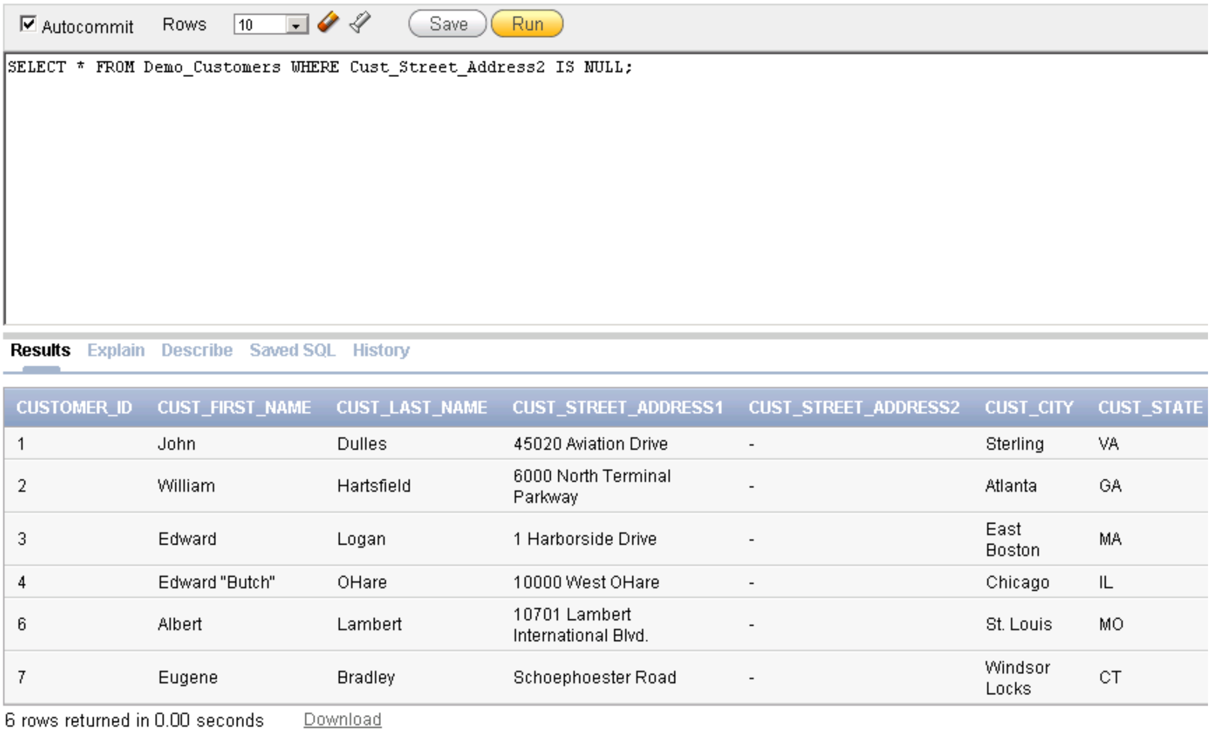
Now if you try:



as you can see there an no results. If you try the following you will get the same problem:



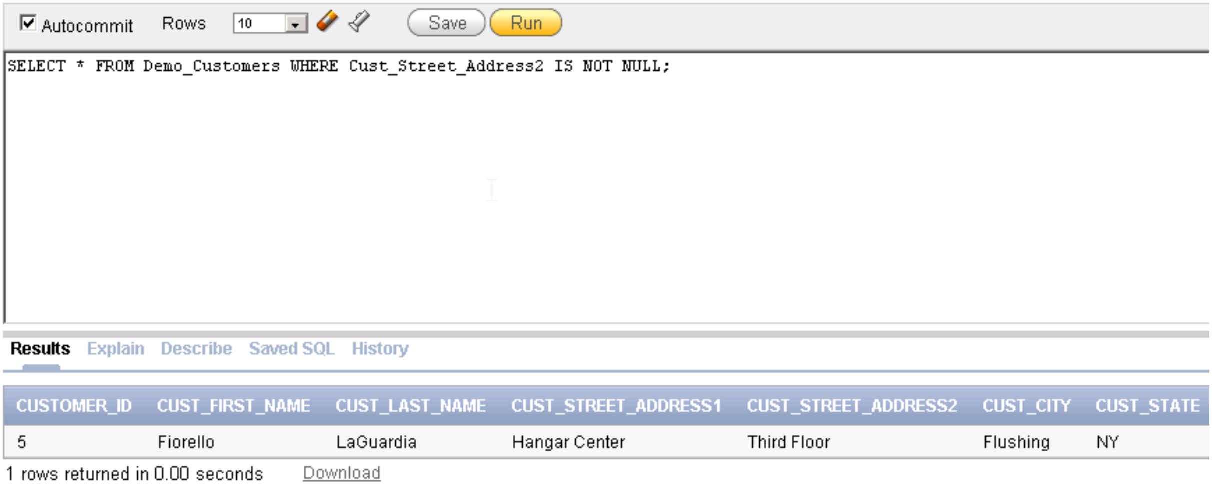
But if you try:



CUSTOMER_ID	CUST_FIRST_NAME	CUST_LAST_NAME	CUST_STREET_ADDRESS1	CUST_STREET_ADDRESS2	CUST_CITY	CUST_STATE
1	John	Dulles	45020 Aviation Drive	-	Sterling	VA
2	William	Hartsfield	6000 North Terminal Parkway	-	Atlanta	GA
3	Edward	Logan	1 Harborside Drive	-	East Boston	MA
4	Edward "Butch"	OHare	10000 West OHare	-	Chicago	IL
6	Albert	Lambert	10701 Lambert International Blvd.	-	St. Louis	MO
7	Eugene	Bradley	Schoephoester Road	-	Windsor Locks	CT

6 rows returned in 0.00 seconds Download

it works! This is the right syntax!


Autocommit
Rows
10
Save
Run
SELECT * FROM Demo_Customers WHERE Cust_Street_Address2 IS NOT NULL;
Results
Explain
Describe
Saved SQL
History
CUSTOMER_ID CUST_FIRST_NAME CUST_LAST_NAME CUST_STREET_ADDRESS1 CUST_STREET_ADDRESS2 CUST_CITY CUST_STATE
5 Fiorello LaGuardia Hangar Center Third Floor Flushing NY
1 rows returned in 0.00 seconds
Download

4.8 SQL AND & OR Operators

The AND operator displays a record if both the first condition AND the second condition are true.

The OR operator displays a record if either the first condition OR the second condition is true.

Remark: And direct XOR doesn't exist actually in SQL! You must use a logical workaround to get it.

The following SQL statement selects all customers from the country "Germany" AND the city "Berlin", in the "Customers" table:

Example with AND:

```
SELECT * FROM Customers
WHERE Country='Germany'
AND City='Berlin';
```

The following SQL statement selects all customers from the city "Berlin" OR "München", in the "Customers" table:

Example with OR:

```
SELECT * FROM Customers
WHERE City='Berlin'
OR City='München';
```

You can also combine AND and OR (use parenthesis to form complex expressions).

The following SQL statement selects all customers from the country "Germany" AND the city must be equal to "Berlin" OR "München", in the "Customers" table:

Example with AND & OR:

```
SELECT * FROM Customers
WHERE Country='Germany'
AND (City='Berlin' OR City='München');
```

4.9 SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set by one or more columns.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in a descending order, you can use the DESC keyword.

```
SELECT column_name,column_name
FROM table_name
ORDER BY column_name,column_name ASC|DESC;
```

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" column:

```
SELECT * FROM Customers
ORDER BY Country;
```

The following SQL statement selects all customers from the "Customers" table, sorted DESCENDING by the "Country" column:

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column:

```
SELECT * FROM Customers
ORDER BY Country,CustomerName;
```

Some DBA write sometimes orders as following:

```
SELECT CustomerName, ContactName, City FROM Customers
ORDER BY 2,3
```

4.10 SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

It is possible to write the INSERT INTO statement in two forms.

The first form does not specify the column names where the data will be inserted, only their values:

```
INSERT INTO table_name
VALUES (value1,value2,value3,...);
```

The second form specifies both the column names and the values to be inserted:

```
INSERT INTO table_name (column1,column2,column3,...)
VALUES (value1,value2,value3,...);
```

We will see late the INSERT ALL statement to insert multiple rows at once!

Assume we wish to insert a new row in the "Customers" table. We can use the following SQL statement:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City,
PostalCode, Country)
VALUES ('Cardinal','Tom B. Erichsen','Skagen
21','Stavanger','4006','Norway');
```

The CustomerID column is automatically updated with a unique number for each record in the table when you use the INSERT INTO statement.

It is also possible to only insert data in specific columns!

The following SQL statement will insert a new row, but only insert data in the "CustomerName", "City", and "Country" columns (and the CustomerID field will of course also be updated automatically):

```
INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

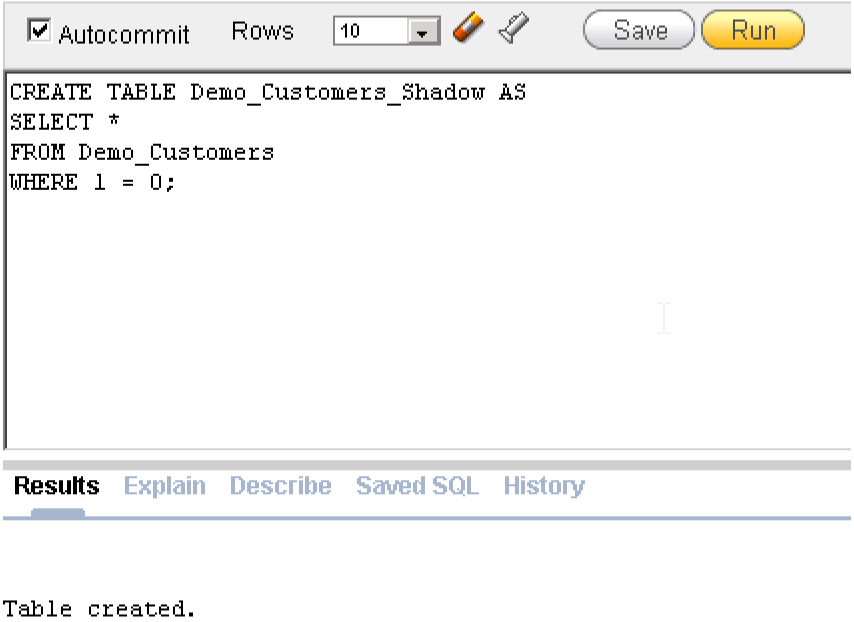
4.10.1 Insert a Null value

To insert a Null value you just have to write the following query:

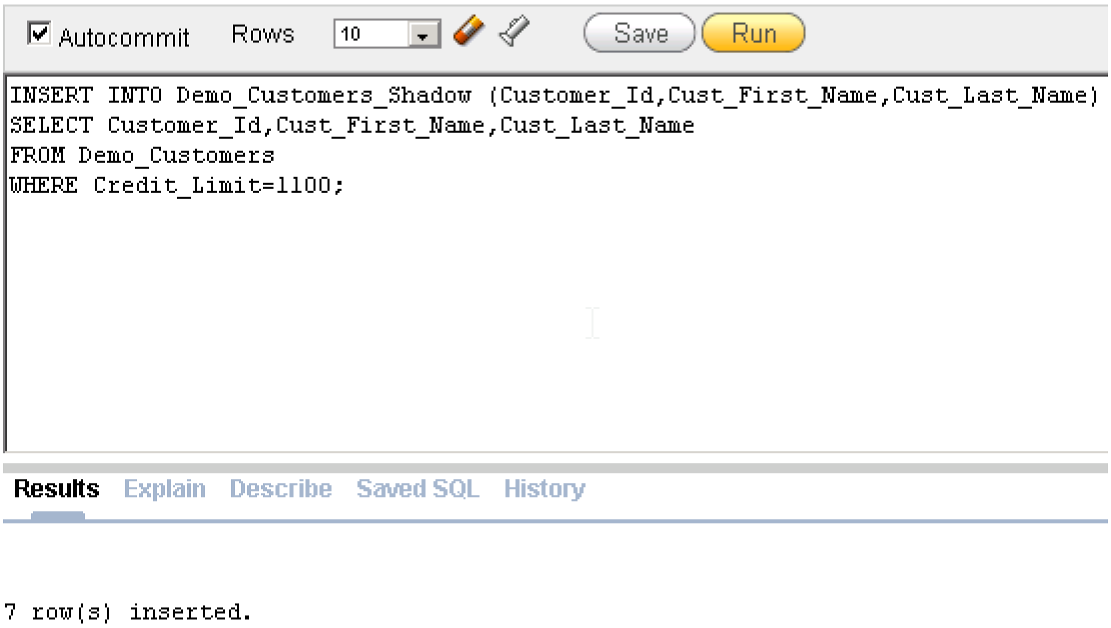
```
INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', Null, 'Norway');
```

4.10.2 Copy the rows of a table into another one

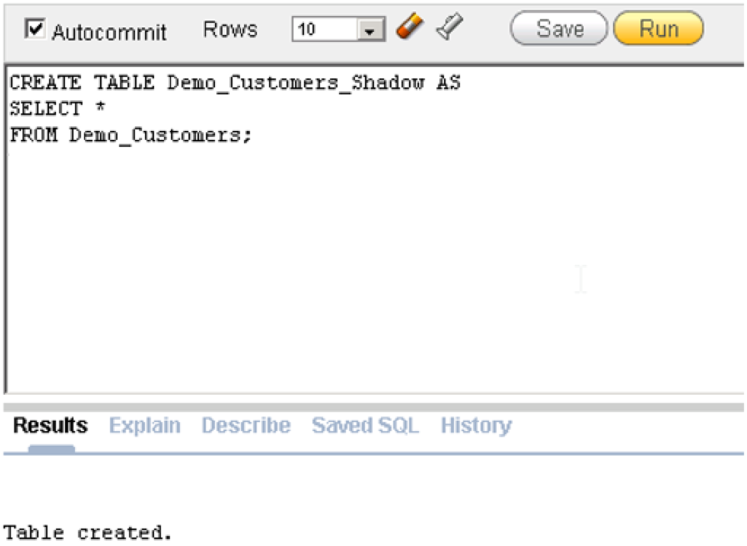
For this example, in Oracle first run the following query that will create a copy of the Demo_Customers table structure:



then you can copy some or all of the rows of the original into the new one:



To create a copy of a table with its data and with its structure then you can simply use:


Autocommit
Rows
10
Save
Run
CREATE TABLE Demo_Customers_Shadow AS
SELECT *
FROM Demo_Customers;
Results
Explain
Describe
Saved SQL
History
Table created.

4.11 SQL UPDATE Statement

The UPDATE statement is used to update existing records in a table.

SQL UPDATE Syntax:

```
UPDATE table_name
SET column1=value1,column2=value2,...
WHERE some_column=some_value;
```

Notice the WHERE clause in the SQL UPDATE statement! The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Assume we wish to update the customer "Alfreds Futterkiste" with a new contact person and city.

We use the following SQL statement:

```
UPDATE Customers
SET ContactName='Alfred Schmidt', City='Hamburg'
WHERE CustomerName='Alfreds Futterkiste';
```

4.12 SQL DELETE Statement

The DELETE statement is used to delete rows in a table.

SQL DELETE Syntax:

```
DELETE FROM table_name
WHERE some_column=some_value;
```

Notice the WHERE clause in the SQL DELETE statement! The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

Assume we wish to delete the customer "Alfreds Futterkiste" from the "Customers" table.

We use the following SQL statement:

```
DELETE FROM Customers
WHERE CustomerName='Alfreds Futterkiste' AND ContactName='Maria Anders';
```

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE * FROM table_name;
```

Note: Be very careful when deleting records. You cannot undo this statement!

4.13 SQL SELECT TOP (and aka BOTTOM) Clause

The SELECT TOP clause is used to specify the number of records to return.

The SELECT TOP clause can be very useful on large tables with thousands of records. Returning a large number of records can impact on performance.

Note: Not all database systems support the SELECT TOP clause.

SQL Server / MS Access Syntax:

```
SELECT TOP number|percent column_name(s)
FROM table_name;
```

Examples on Microsoft Access:

Products that selects the two first records from the "Customers" table

```
SELECT TOP 2 * FROM Customers;
```

or with percent selects the first 50% of the records from the "Customers" table:

```
SELECT TOP 50 PERCENT * FROM Customers;
```

MySQL Syntax:

```
SELECT column_name(s)
FROM table_name
LIMIT number;
```

Example MySQL Syntax:

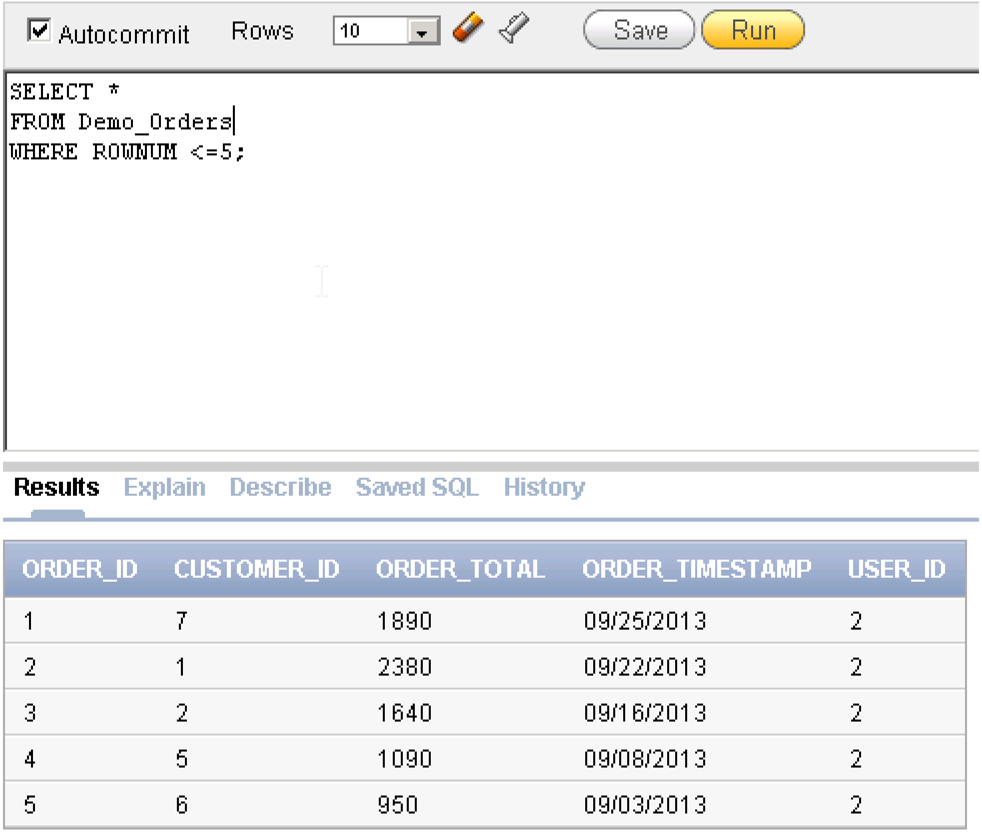
```
SELECT *
FROM Persons
LIMIT 5;
```

Oracle Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE ROWNUM <= number;
```

Example Oracle Syntax:

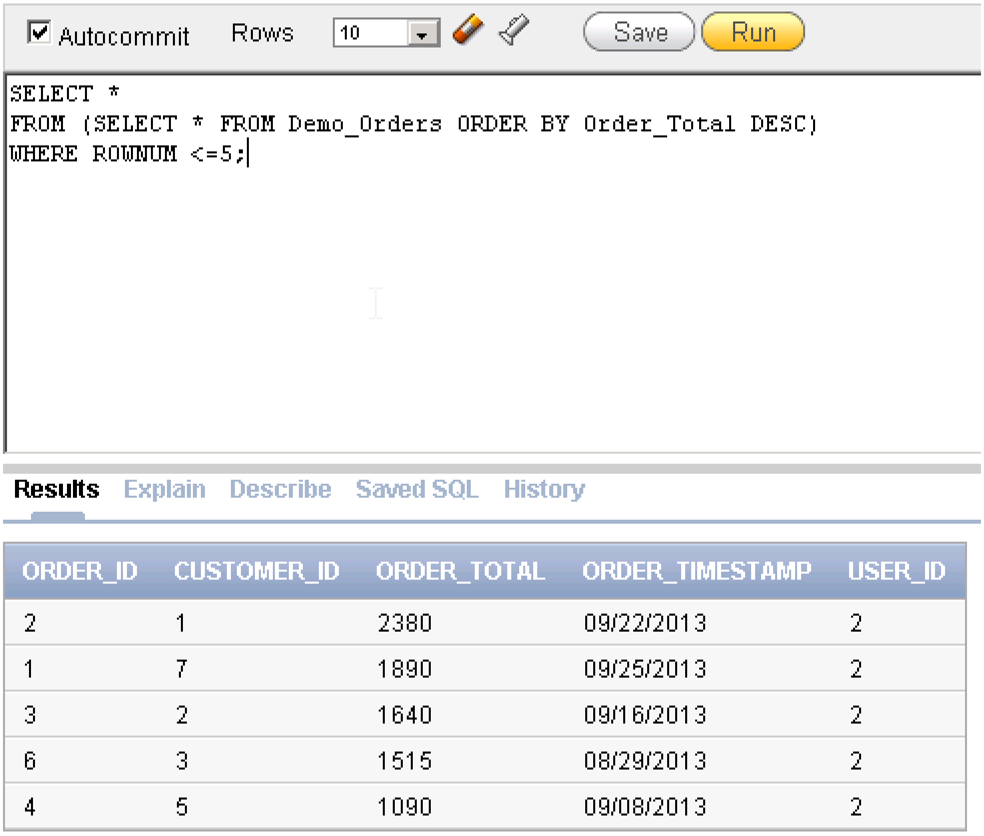
Five first orders:




```
SELECT *
FROM Demo_Orders
WHERE ROWNUM <=5;
```

ORDER_ID	CUSTOMER_ID	ORDER_TOTAL	ORDER_TIMESTAMP	USER_ID
1	7	1890	09/25/2013	2
2	1	2380	09/22/2013	2
3	2	1640	09/16/2013	2
4	5	1090	09/08/2013	2
5	6	950	09/03/2013	2

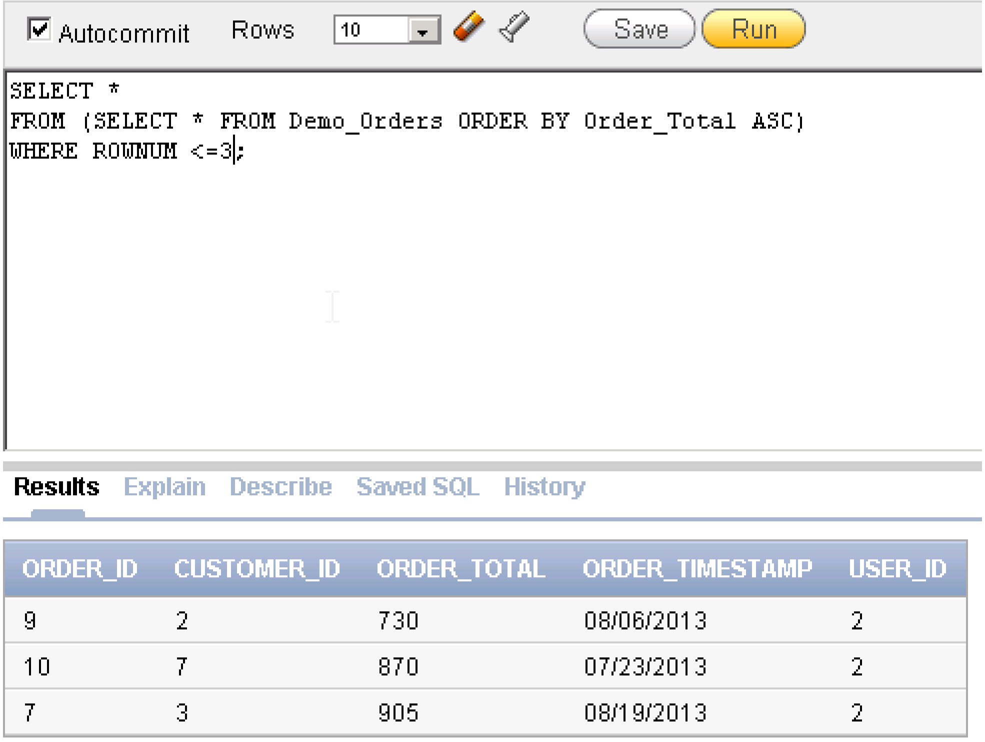
Five highest orders:




```
SELECT *
FROM (SELECT * FROM Demo_Orders ORDER BY Order_Total DESC)
WHERE ROWNUM <=5;
```

ORDER_ID	CUSTOMER_ID	ORDER_TOTAL	ORDER_TIMESTAMP	USER_ID
2	1	2380	09/22/2013	2
1	7	1890	09/25/2013	2
3	2	1640	09/16/2013	2
6	3	1515	08/29/2013	2
4	5	1090	09/08/2013	2

Three smallest orders:




```
SELECT *
FROM (SELECT * FROM Demo_Orders ORDER BY Order_Total ASC)
WHERE ROWNUM <=3;
```

ORDER_ID	CUSTOMER_ID	ORDER_TOTAL	ORDER_TIMESTAMP	USER_ID
9	2	730	08/06/2013	2
10	7	870	07/23/2013	2
7	3	905	08/19/2013	2

4.14 SQL LIKE Operator

The LIKE operator is used to search for a specified pattern in a column.

SQL LIKE Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern;
```

The following SQL statement selects all customers with a City starting with the letter "s":

```
SELECT * FROM Customers
WHERE City LIKE 's%';
```

or on MS Access:

```
SELECT * FROM Customers
WHERE City LIKE 's*';
```

The following SQL statement selects all customers with a Country containing the pattern "land":

```
SELECT * FROM Customers
WHERE Country LIKE '%land%';
```

Using the NOT keyword allows you to select records that does NOT match the pattern.

The following SQL statement selects all customers with a Country NOT containing the pattern "land":

```
SELECT * FROM Customers
WHERE Country NOT LIKE '%land%';
```

4.14.1 SQL Wildcards

In SQL, wildcard characters are used with the SQL LIKE operator.

With official SQL, the wildcards are:

Wildcard	Description
%	A substitute for zero or more characters
_	A substitute for a single character
[*charlist*]	Sets and ranges of characters to match
[^*charlist*] or [!*charlist*]	Matches only a character NOT specified within the brackets

Tableau 4 Common SQL Wildcards

The following SQL statement selects all customers with a City starting with any character, followed by "erlin":

```
SELECT * FROM Customers
WHERE City LIKE '_erlin';
```

In MS Access the following Statement work for only one letter (but that's not standard SQL):

```
SELECT * FROM Customers
WHERE City LIKE 's?o paulo';
```

The following SQL statement selects all customers with a City starting with "b", "s", or "p":

```
SELECT * FROM Customers
WHERE City LIKE '[bsp]%';
```

The following SQL statement selects all customers with a City starting with "a", "b", or "c":

```
SELECT * FROM Customers
WHERE City LIKE '[a-c]%';
```

The following SQL statement selects all customers with a City NOT starting with "b", "s", or "p":

```
SELECT * FROM Customers
WHERE City LIKE '[!bsp]%';
```

Or the equivalent:

```
SELECT * FROM Customers
WHERE City NOT LIKE '[bsp]%';
```

4.14.2 SQL REGEX

With the REGEX option in MySQL, the following query:

```
SELECT * FROM Customers
WHERE City LIKE 's%';
```

Will be written:

```
SELECT * FROM Customers
WHERE City REGEX '^s';
```

And:

```
SELECT * FROM Customers
WHERE City LIKE '[a-c]%';
```

Will be written:

```
SELECT * FROM Customers
WHERE City LIKE '^[a-c]';
```

The reste is about a REGEX training…

4.15 SQL IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

SQL IN Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...);
```

The following SQL statement selects all customers with a City of "Paris" or "London":

```
SELECT * FROM Customers
WHERE City IN ('Paris','London');
```

MS Access will write the same automatically as following (but previous syntax will still work)...:

```
SELECT * FROM Customers
WHERE (City="Alain") OR (City="Albert");
```

4.16 SQL BETWEEN and NOT BETWEEN Operators

The BETWEEN operator selects values within a range. The values can be numbers, text, or dates.

SQL BETWEEN Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

The following SQL statement selects all products with a price BETWEEN 10 and 20:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

To display the products outside the range of the previous example, use NOT BETWEEN:

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

The following SQL statement selects all products with a price BETWEEN 10 and 20, but products with a CategoryID of 1,2, or 3 should not be displayed:

```
SELECT * FROM Products
WHERE (Price BETWEEN 10 AND 20)
AND NOT CategoryID IN (1,2,3);
```

The following SQL statement selects all products with a ProductName beginning with any of the letter BETWEEN 'C' and 'M':

```
SELECT * FROM Products
WHERE ProductName BETWEEN 'C' AND 'M';
```

The following SQL statement selects all products with a ProductName beginning with any of the letter NOT BETWEEN 'C' and 'M':

```
SELECT * FROM Products
WHERE ProductName NOT BETWEEN 'C' AND 'M';
```

The following SQL statement selects all orders with an OrderDate BETWEEN '04-July-1996' and '09-July-1996':

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN #07/04/1996# AND #07/09/1996#;
```

4.17 SQL Cartesian Product

What do you think happens if you run:

```
SELECT c.CustomerName, c.CustomerID, o.OrderID
FROM Customers c, Orders o
WHERE c.CustomerID=5;
```

You will then have de cartesian product of all the combinations... for sure this is not what you are expecting... Then see what's next about JOIN operator.

4.18 SQL JOIN

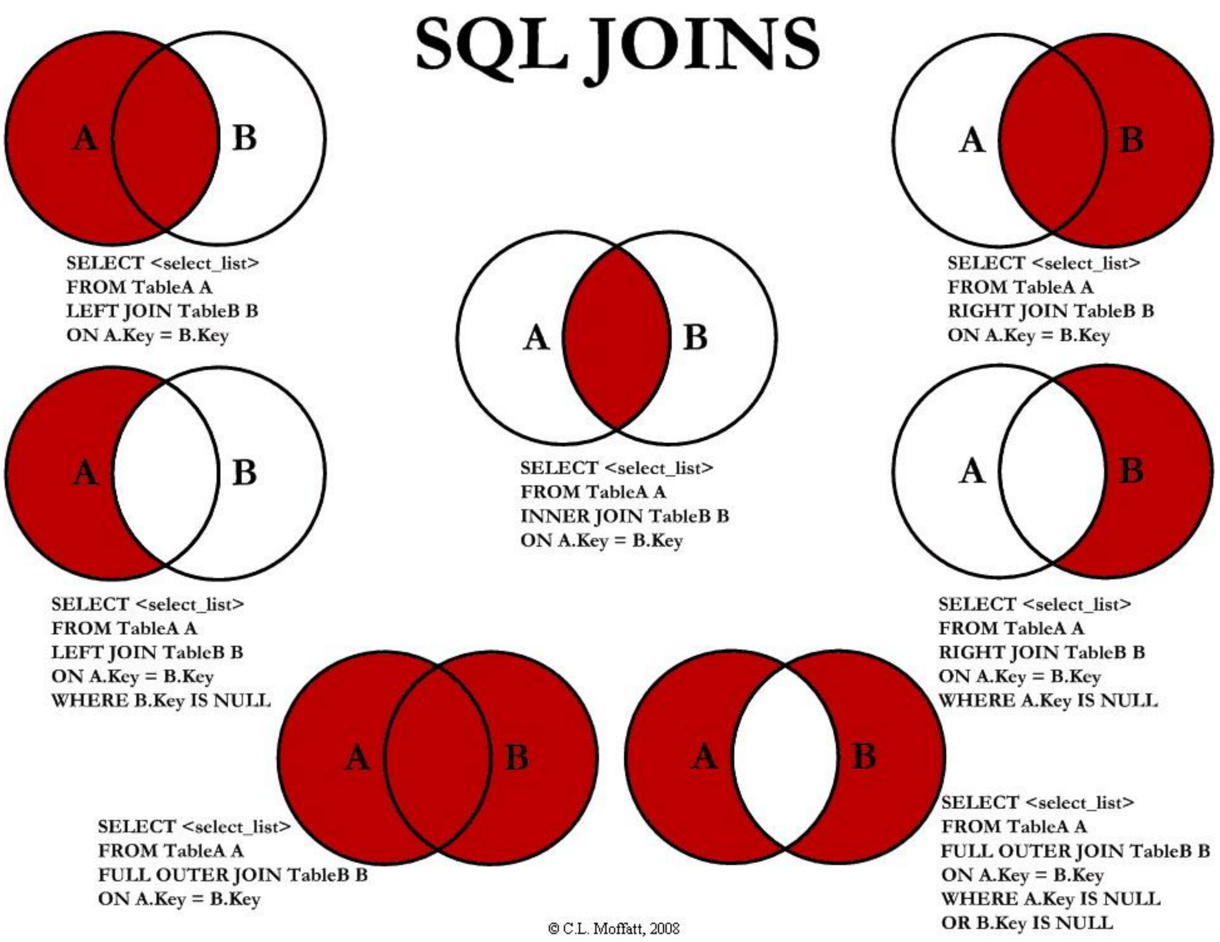


Figure 2Illustrated Common SQL Joins

4.18.1 SQL INNER JOIN statement

4.18.1.1 INNER JOIN with 2 tables

The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns in both tables.

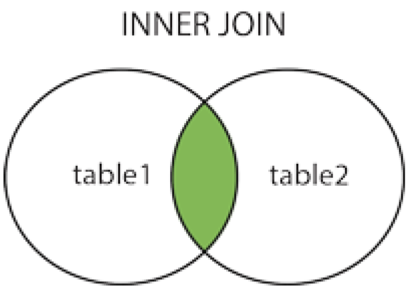
SQL INNER JOIN Syntax:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name=table2.column_name;
```

or:

```
SELECT column_name(s)
FROM table1
JOIN table2
ON table1.column_name=table2.column_name;
```

PS! INNER JOIN is the same as JOIN.



Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
...						

And a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2
...				

The following SQL statement will return all customers with orders:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

and compare this query with the example of the cartesian product:

```
SELECT Customers.CustomerName, Customers.CustomerID, Orders.OrderID
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
WHERE Customers.CustomerID=5;
```

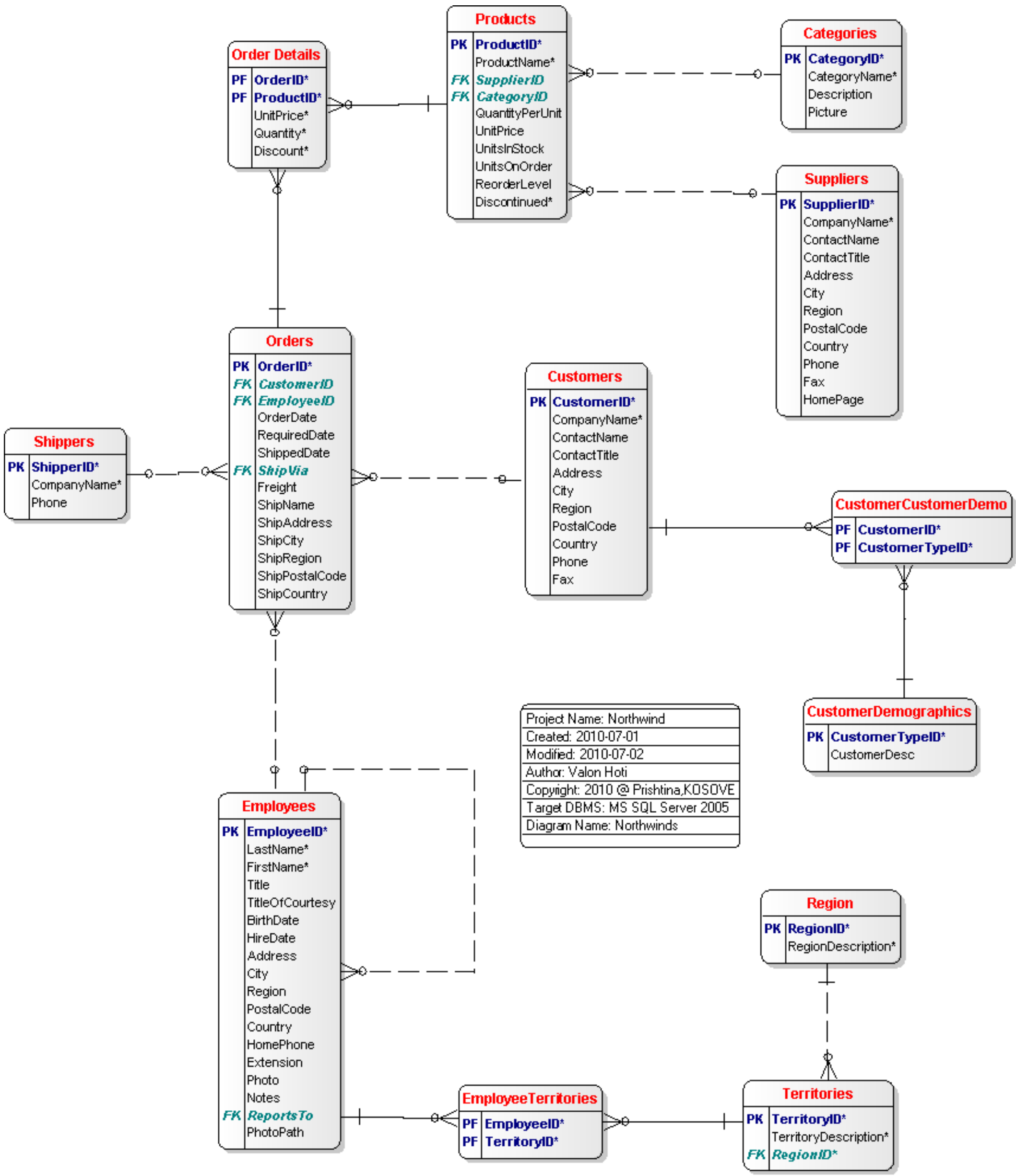
4.18.1.2 INNER JOIN with 4 tables

This is a very important example to understand!!!!

The following SQL statement will return all customers with orders and the saler name:

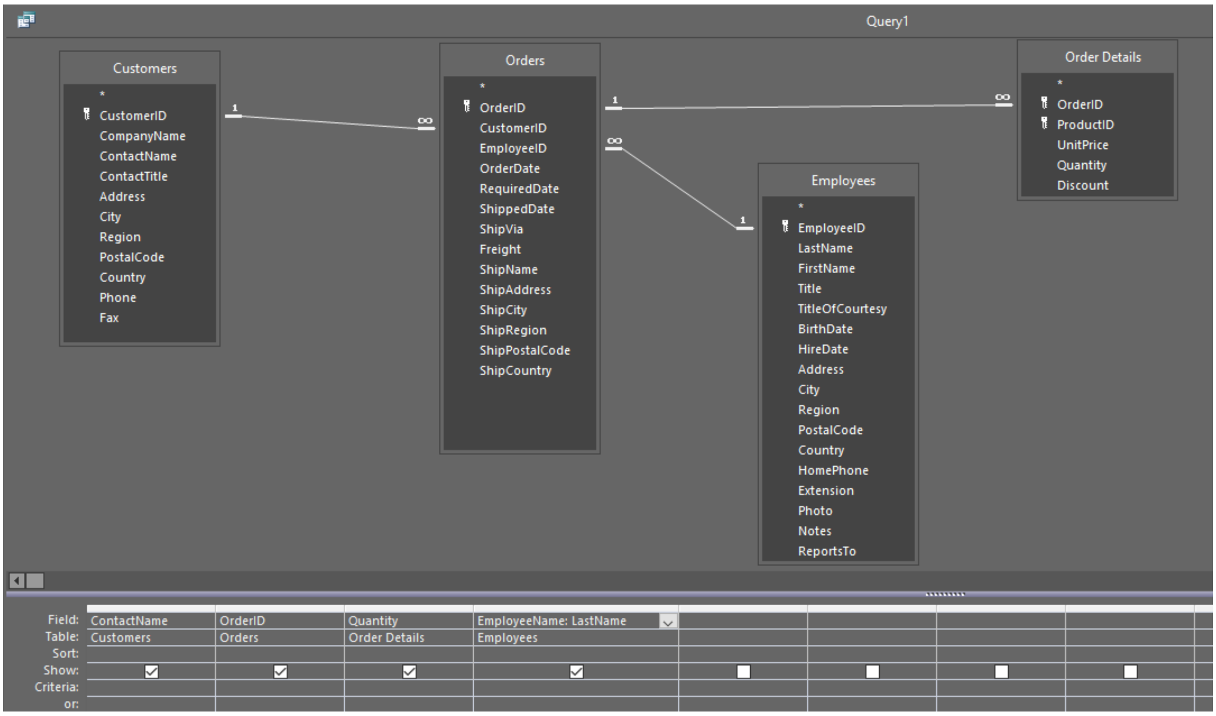
```
SELECT Customers.CustomerName, Orders.OrderId, OrderDetails.Quantity,
Employees.LastName As EmployeeName
FROM Customers
    INNER JOIN Orders
        ON Customers.CustomerID = Orders.CustomerID
    INNER JOIN OrderDetails
        ON OrderDetails.OrderID= Orders.OrderID
    INNER JOIN Employees
        ON Employees.EmployeeID= Orders.EmployeeID;
```

To understand the following shema will help again:



We will use this query late for the study of CROSS JOIN statement.

For information the equivalent in Microsoft Access looks like following:



And the corresponding automated generated SQL:

```
SELECT Customers.ContactName, Orders.OrderID, [Order Details].Quantity, Employees.LastName AS EmployeeName
FROM (Employees INNER JOIN (Customers INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID)
ON Employees.EmployeeID = Orders.EmployeeID) INNER JOIN [Order Details] ON Orders.OrderID = [Order Details].OrderID;
```

Formatted a little bit this gives:

```
SELECT Customers.ContactName, Orders.OrderID, [Order Details].Quantity, Employees.LastName
FROM (Employees
  INNER JOIN (Customers
  INNER JOIN Orders
    ON Customers.CustomerID = Orders.CustomerID)
    ON Employees.EmployeeID = Orders.EmployeeID)
  INNER JOIN [Order Details]
    ON Orders.OrderID = [Order Details].OrderID;
```

It can be seen that the SQL generated by Microsoft Access is far from ideal ... Even if by copying straight into MySQL or other this code works perfectly (just need to adapt the name of one table and one of the fields!).

By cons the opposite does not apply! Copying the given SQL at the beginning in Microsoft Access will not work (even by adapting small differences in names) !!!

4.18.2 SQL LEFT JOIN statement (OUTER JOIN Family)

The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.

Remarks: Starting with Oracle9i, the confusing outer join syntax using the '(+)' notation has been superseded by ISO 1999 outer join syntax. As we know, there are three types of outer joins, left, right, and full outer join. The purpose of an outer join is to include non-matching rows, and the outer join returns these missing columns as NULL values.

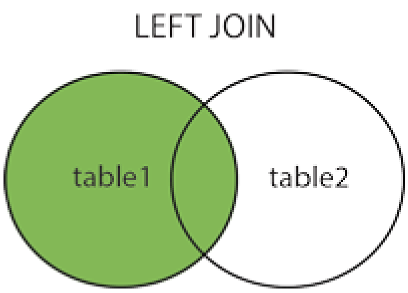
SQL LEFT JOIN Syntax:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name=table2.column_name;
```

or:

```
SELECT column_name(s)
FROM table1
LEFT OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

PS! In some databases LEFT JOIN is called **LEFT OUTER JOIN**.



Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
...						

And a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2
...				

The following SQL statement will return all customers, and any orders they might have:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

The LEFT JOIN keyword will then return all the rows from the left table (Customers), even if there are no matches in the right table (Orders)

CustnomerName	OrderID
Alfreds Futterkiste	null
Ana Trujillo Emparedados y helados	10308
Antonio Moreno Taquería	10365
Around the Horn	10355
Around the Horn	10383
B's Beverages	10289
Berglunds snabbköp	10278
Berglunds snabbköp	10280
Berglunds snabbköp	10384
Blauer See Delikatessen	null
...	

4.18.3 SQL RIGHT JOIN statement (OUTER JOIN FAMILY)

The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

SQL RIGHT JOIN Syntax:

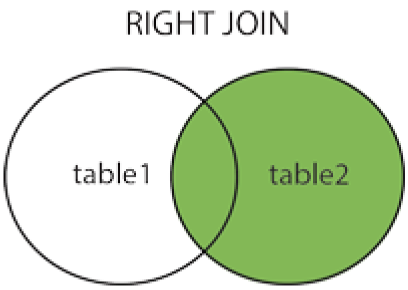
```
SELECT column_name(s)
FROM table1
```

```
RIGHT JOIN table2
ON table1.column_name=table2.column_name;
```

or:

```
SELECT column_name(s)
FROM table1
RIGHT OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

PS! In some databases RIGHT JOIN is called **RIGHT OUTER JOIN**.



Below is a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2
...				

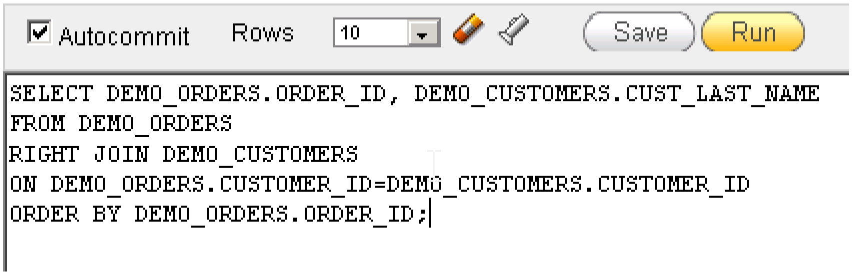
And a selection from the "Employees" table:

EmployeeID	LastName	FirstName	BirthDate	Photo	Notes
1	Davolio	Nancy	12/8/1968	EmpID1.pic	Education includes a BA in psychology.....
2	Fuller	Andrew	2/19/1952	EmpID2.pic	Andrew received his BTS commercial and....
3	Leverling	Janet	8/30/1963	EmpID3.pic	Janet has a BS degree in chemistry....
...					

The following SQL statement will return all employees, and any orders they have sell:

```
SELECT Orders.OrderID, Employees.FirstName
FROM Orders
RIGHT JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID
ORDER BY Orders.OrderID;
```

For the exemple database of ORACLE Server this will be:



or its equivalent (but less intuitive to read):

```
SELECT Orders.OrderID, Employees.FirstName
FROM Employees
LEFT JOIN Orders
ON Orders.EmployeeID=Employees.EmployeeID
ORDER BY Orders.OrderID;
```

This will return:

OrderID	FirstName
	Adam
10248	Steven
10249	Michael
10250	Margaret
10251	Janet
...	

As you can see Adam did never sell anything but is still visible. Try now:

```
SELECT Orders.OrderID, Employees.FirstName
FROM Orders
LEFT JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID
ORDER BY Orders.OrderID;
```

and you will see that you have then only employees that did sell something (Adam will not be visible anymore).

You seem to be asking, "If I can rewrite a RIGHT OUTER JOIN using LEFT OUTER JOIN syntax then why have a RIGHT OUTER JOIN syntax at all?" I think the answer to this question is, because the designers of the language didn't want to place such a restriction on users (and I think they would have been criticized if they did), which would force users to change the order of tables in the FROM clause in some circumstances when merely changing the join type.

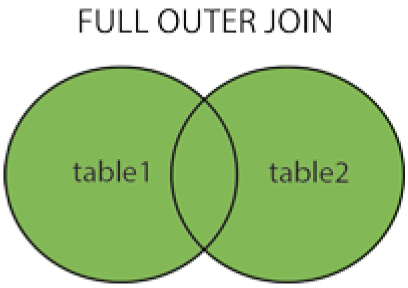
4.18.4 SQL FULL OUTER JOIN statement (OUTER JOIN FAMILY)

The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).

The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

SQL FULL OUTER JOIN Syntax:

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name=table2.column_name;
```



MySQL & Microsoft Access lacks support for FULL OUTER JOIN!!!

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
...						

And a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

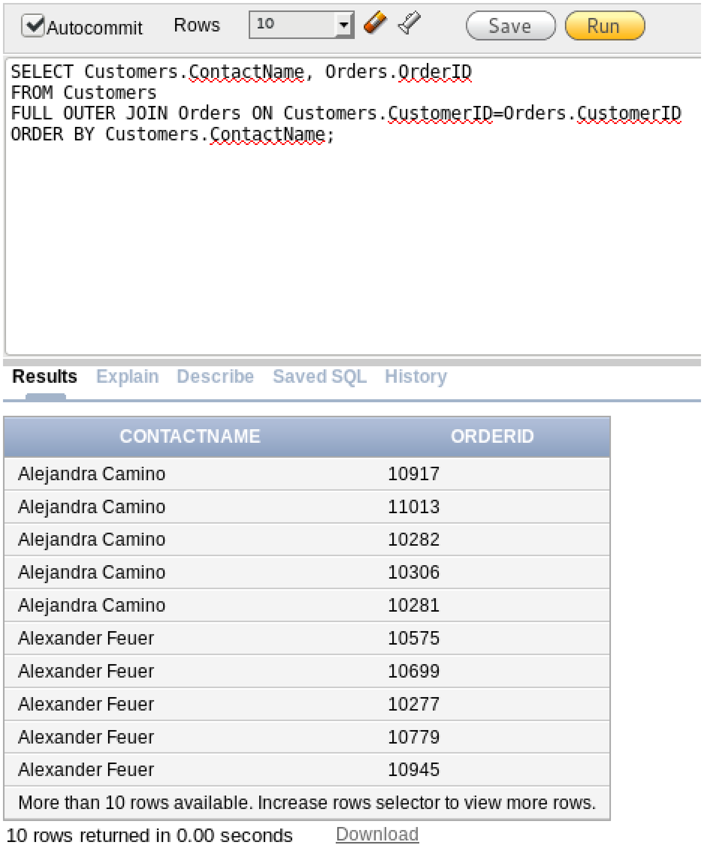
The following SQL statement selects all customers, and all orders:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

that will result in on the W3Schools website:

CustomerName	OrderID
Alfreds Futterkiste	
Ana Trujillo Emparedados y helados	10308
Antonio Moreno Taquería	10365
	10382
	10351
...	

In Oracle it will give you:




```
SELECT Customers.ContactName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.ContactName;
```

CONTACTNAME	ORDERID
Alejandra Camino	10917
Alejandra Camino	11013
Alejandra Camino	10282
Alejandra Camino	10306
Alejandra Camino	10281
Alexander Feuer	10575
Alexander Feuer	10699
Alexander Feuer	10277
Alexander Feuer	10779
Alexander Feuer	10945
More than 10 rows available. Increase rows selector to view more rows.	

10 rows returned in 0.00 seconds Download

To do the same in mySQL you will need (enjoy not being on Oracle)...:

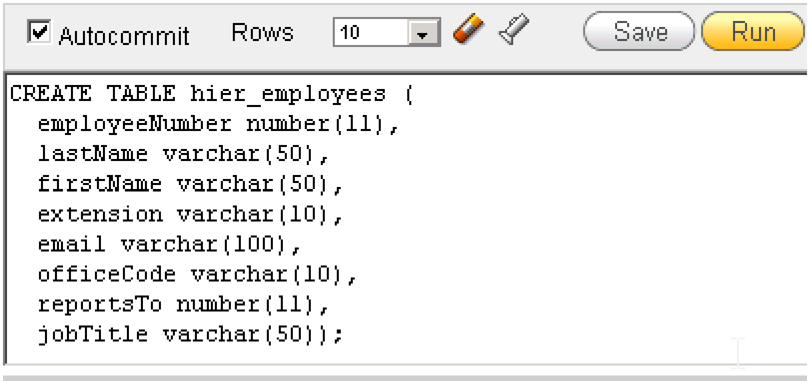
```
SELECT   Customers.CustomerName, Orders.OrderID
FROM     Customers
LEFT JOIN
         Orders
ON       Customers.CustomerID=Orders.CustomerID
;
UNION ALL
SELECT   NULL, OrderID
FROM     orders
WHERE    OrderID NOT IN
         (
         SELECT   CustomerID
         FROM     Customers
         );
```

I give you imagine how to deal with multiple FULL JOINS in mySQL looks like…

4.18.5 SQL SELF JOIN (circular join) like syntax

While self-joins (also named "circular join" or "auto join") rarely are used on a normalized database, you can use them to reduce the number of queries that you execute when you compare values of different columns of the same table.

For this example, first create the following table structure in Oracle:



and insert following datas (with the horrible Oracle Syntax):

```
INSERT ALL
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1002,'Murphy','Diane','x5800','dmurphy@classicmodelcars.com','1',NULL,'President')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1056,'Patterson','Mary','x4611','mpatterso@classicmodelcars.com','1',1002,'VP Sales')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1076,'Firrelli','Jeff','x9273','jfirrelli@classicmodelcars.com','1',1002,'VP Marketing')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1088,'Patterson','William','x4871','wpatterson@classicmodelcars.com','6',1056,'Sales Manager (APAC)')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1102,'Bondur','Gerard','x5408','gbondur@classicmodelcars.com','4',1056,'Sale Manager (EMEA)')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1143,'Bow','Anthony','x5428','abow@classicmodelcars.com','1',1056,'Sales Manager (NA)')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1165,'Jennings','Leslie','x3291','ljennings@classicmodelcars.com','1',1143,'Sales Rep')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1166,'Thompson','Leslie','x4065','lthompson@classicmodelcars.com','1',1143,'Sales Rep')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1188,'Firrelli','Julie','x2173','jfirrelli@classicmodelcars.com','2',1143,'Sales Rep')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1216,'Patterson','Steve','x4334','spatterson@classicmodelcars.com','2',1143,'Sales Rep')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1286,'Tseng','Foon Yue','x2248','ftseng@classicmodelcars.com','3',1143,'Sales Rep')
```

```
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1323,'Vanauf','George','x4102','gvanauf@classicmodelcars.com','3',1143,'Sales Rep')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1337,'Bondur','Loui','x6493','lbondur@classicmodelcars.com','4',1102,'Sales Rep')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1370,'Hernandez','Gerard','x2028','ghernande@classicmodelcars.com','4',1102,'Sales Rep')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1401,'Castillo','Pamela','x2759','pcastillo@classicmodelcars.com','4',1102,'Sales Rep')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1501,'Bott','Larry','x2311','lbott@classicmodelcars.com','7',1102,'Sales Rep')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1504,'Jones','Barry','x102','bjones@classicmodelcars.com','7',1102,'Sales Rep')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1611,'Fixter','Andy','x101','afixter@classicmodelcars.com','6',1088,'Sales Rep')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1612,'Marsh','Peter','x102','pmarsh@classicmodelcars.com','6',1088,'Sales Rep')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1619,'King','Tom','x103','tking@classicmodelcars.com','6',1088,'Sales Rep')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1621,'Nishi','Mami','x101','mnishi@classicmodelcars.com','5',1056,'Sales Rep')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1625,'Kato','Yoshimi','x102','ykato@classicmodelcars.com','5',1621,'Sales Rep')
INTO hier_employees(employeeNumber,lastName,firstName,extension,email,officeCode,reportsTo,jobTitle) values
(1702,'Gerard','Martin','x2312','mgerard@classicmodelcars.com','4',1102,'Sales Rep')
SELECT * FROM dual;
```

You will have something like this:

EDIT	EMPLOYEENUMBER	LASTNAME	FIRSTNAME	EXTENSION	EMAIL	OFFICECODE	REPORTSTO	JOBTITLE
	1002	Murphy	Diane	x5800	dmurphy@classicmodelcars.com	1	-	President
	1056	Patterson	Mary	x4611	mpatterso@classicmodelcars.com	1	1002	VP Sales
	1076	Firrelli	Jeff	x9273	jfirrelli@classicmodelcars.com	1	1002	VP Marketing
	1088	Patterson	William	x4871	wpatterson@classicmodelcars.com	6	1056	Sales Manager (APAC)
	1102	Bondur	Gerard	x5408	gbondur@classicmodelcars.com	4	1056	Sale Manager (EMEA)
	1143	Bow	Anthony	x5428	abow@classicmodelcars.com	1	1056	Sales Manager (NA)
	1165	Jennings	Leslie	x3291	ljennings@classicmodelcars.com	1	1143	Sales Rep
	1166	Thompson	Leslie	x4065	lthompson@classicmodelcars.com	1	1143	Sales Rep
	1188	Firrelli	Julie	x2173	jfirrelli@classicmodelcars.com	2	1143	Sales Rep
	1216	Patterson	Steve	x4334	spatterson@classicmodelcars.com	2	1143	Sales Rep
	1286	Tseng	Foon Yue	x2248	ftseng@classicmodelcars.com	3	1143	Sales Rep
	1323	Vanauf	George	x4102	gvanauf@classicmodelcars.com	3	1143	Sales Rep
	1337	Bondur	Loui	x6493	lbondur@classicmodelcars.com	4	1102	Sales Rep
	1370	Hernandez	Gerard	x2028	ghernande@classicmodelcars.com	4	1102	Sales Rep
	1401	Castillo	Pamela	x2759	pcastillo@classicmodelcars.com	4	1102	Sales Rep

row(s) 1 - 15 of 23

In the employees table, we store not only employee's data but also organization structure data. The REPORTSTO column is used to determine the manager ID of an employee.

In order to get the whole organization structure, we can join the HIER_EMPLOYEES table to itself using the EMPLOYEENUMBER and REPORTSTO columns.

Autocommit Rows 10 Save Run

```
SELECT E1.LASTNAME AS EMPLOYEE, E2.LASTNAME AS MANAGER
FROM hier_employees E1, hier_employees E2
WHERE E1.REPORTSTO=E2.EMPLOYEENUMBER;
```

that will result in:

EMPLOYEE	MANAGER
Firrelli	Murphy
Patterson	Murphy
Nishi	Patterson
Bow	Patterson
Bondur	Patterson
Patterson	Patterson
King	Patterson
Marsh	Patterson
Fixter	Patterson
Gerard	Bondur
Jones	Bondur
Bott	Bondur
Castillo	Bondur
Hernandez	Bondur
Bondur	Bondur
Vanauf	Bow
Tseng	Bow
Patterson	Bow
Firrelli	Bow
Thompson	Bow
Jennings	Bow
Kato	Nishi

But **the Top Manager is missing**... Using the following syntax:

```
SELECT m.LASTNAME AS EMPLOYEE, e.LASTNAME AS MANAGER
FROM hier_employees e
LEFT JOIN hier_employees m ON m.EMPLOYEENUMBER=e.REPORTSTO;
```

we get the improved result (now shown with all rows):

EMPLOYEE	MANAGER
Murphy	Firrelli
Murphy	Patterson
Patterson	Nishi
Patterson	Bow
Patterson	Bondur
Patterson	Patterson
Patterson	King
Patterson	Marsh
Patterson	Fixter
Bondur	Gerard
Bondur	Jones
Bondur	Bott
Bondur	Castillo
Bondur	Hernandez
Bondur	Bondur
Bow	Vanauf
Bow	Tseng
Bow	Patterson
Bow	Firrelli
Bow	Thompson
Bow	Jennings
Nishi	Kato
-	Murphy

4.18.5.1 SQL CONNECT BY hierarchical queries

If a table contains hierarchical data, then you can select rows in a hierarchical order using the hierarchical query clause.

This is especially useful for:

- **Orgcharts!**
- **Project Gantt Plannings!**
- **MindMaps!**
- **Forum threads!**
- **...**

Consider the following table for a basic example:

EMP

Table | **Data** | Indexes | Model | Constraints | Grants | Statistics | UI Defaults | Triggers | Dependencies | SQL

Query | Count Rows | Insert Row

EDIT	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
	7839	KING	PRESIDENT	-	11/17/1981	5000	-	10
	7698	BLAKE	MANAGER	7839	05/01/1981	2850	-	30
	7782	CLARK	MANAGER	7839	06/09/1981	2450	-	10
	7566	JONES	MANAGER	7839	04/02/1981	2975	-	20
	7788	SCOTT	ANALYST	7566	12/09/1982	3000	-	20
	7902	FORD	ANALYST	7566	12/03/1981	3000	-	20
	7369	SMITH	CLERK	7902	12/17/1980	800	-	20
	7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
	7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
	7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
	7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
	7876	ADAMS	CLERK	7788	01/12/1983	1100	-	20
	7900	JAMES	CLERK	7698	12/03/1981	950	-	30
	7934	MILLER	CLERK	7782	01/23/1982	1300	-	10

row(s) 1 - 14 of 14

The following query will create the **complete** structure of employees from the president to the bottom down employee:

Autocommit Rows 20 Save Run

```
SELECT e.ename  Organigramme
FROM emp E
START WITH e.mgr IS NULL
connect by E.MGR = prior E.EMPNO;
```

Results Explain Describe Saved SQL History

ORGANIGRAMME
KING
JONES
SCOTT
ADAMS
FORD
SMITH
BLAKE
ALLEN
WARD
MARTIN
TURNER
JAMES
CLARK
MILLER

or in a prettier way:

Autocommit Rows 20 Save Run

```
SELECT LPAD(E.EName,LENGTH(E.EName)+(LEVEL-1)*3,'+') "Horizontal Orgchart"
FROM Emp E
WHERE E.EName <> 'BLAKE'
START WITH E.Mgr IS NULL
CONNECT BY E.Mgr = PRIOR E.EmpNo;
```

SELECT LPAD(ENAME, LENGTH(ENAME)+(LEVEL-1)*3,'+') "Horizontoal Orgchart" FROM mydata WHERE ENAME<>'BLAKE' START WITH MGR IS NULL CONNECT BY MGR=PRIOR EMPNO

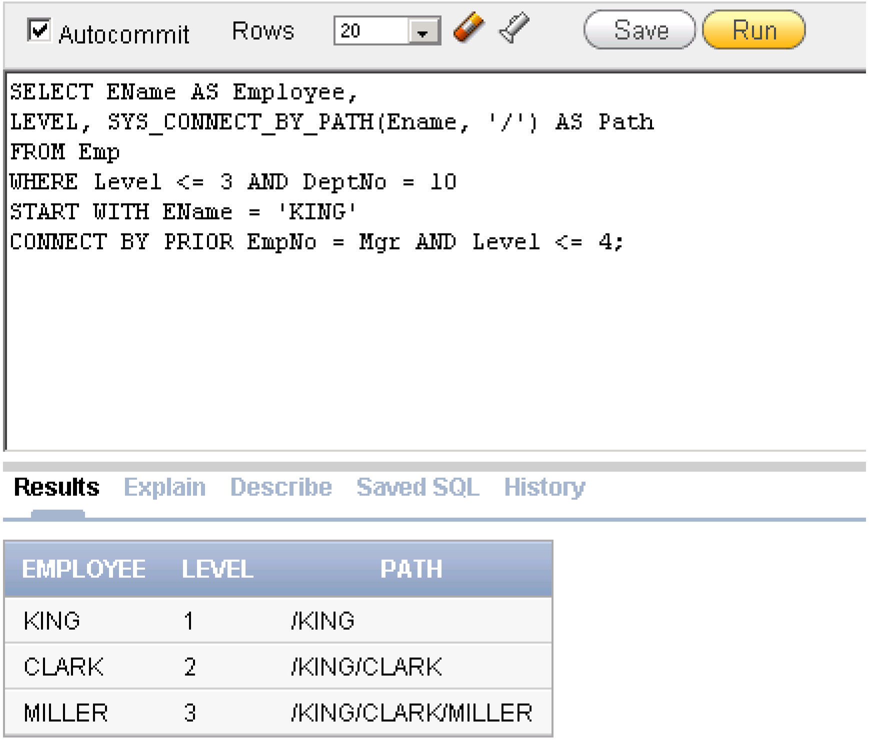
Horizontal Orgchart
KING
+++JONES
++++++SCOTT
+++++++++ADAMS
++++++FORD
+++++++++SMITH
++++++ALLEN
++++++WARD
++++++MARTIN
++++++TURNER
++++++JAMES
+++CLARK
++++++MILLER

or for a partial orgchart:

```
SELECT LPAD(E.EName,LENGTH(E.EName)+(LEVEL-1)*3,'+') "Horizontal Orgchart"
FROM Emp E
START WITH E.EName = 'ADAMS'
CONNECT BY E.EmpNo = PRIOR E.Mgr;
```

Horizontal Orgchart
ADAMS
+++SCOTT
++++++JONES
+++++++++KING

or another mor complicated way:



they are other CONNECT BY statement available in Oracle... for more see on Google.

4.18.6 SQL CROSS JOIN syntax

The following **plain cross query** returns all possible combinations of Customers and Suppliers (then the total number of rows of the result will be the multiplication of the rows of the two, three, ... tables used for the query):

```
SELECT CustomerName, ShipperName FROM Customers CROSS Join Shippers
```

This is equivalent to the ANSI SQL:1989 syntax:

```
SELECT CustomerName, ShipperName
FROM Customers, Shippers
```

Remark: CROSS JOIN is not available in MS Access

You won't found very interesting example of this query in books for non-statisticians but remember that we saw in the Stastics coures how to proceed to a **chi-2 test of independence** using a cross table and this is the case where such query can be very useful to link the resulting view to a statistical software.

This type of query can also be used to generate a table with a combinations of vendors names and sales dates to make statistical forecasting for each vendor with all existing dates (see Quantitative Finance course).

For an example consider first the following query on the W3 School website:

```
SELECT Customers.CustomerName, Employees.LastName As EmployeeName,
Sum(OrderDetails.Quantity)AS SumQuantity
FROM Customers
    INNER JOIN Employees
```

```
        ON Employees.EmployeeID=  Orders.EmployeeID
     INNER JOIN OrderDetails
        ON OrderDetails.OrderID= Orders.OrderID
    INNER JOIN Orders
        ON Customers.CustomerID = Orders.CustomerID
GROUP BY Customers.CustomerName, Employees.LastName
ORDER BY Customers.CustomerName;
```

This will give:

CustomerName	EmployeeName	SumQuantity
Ana Trujillo Emparedados y helados	King	6
Antonio Moreno Taquería	Leverling	24
Around the Horn	Callahan	55
Around the Horn	Suyama	50
B's Beverages	King	39
Berglunds snabbköp	Callahan	64
Berglunds snabbköp	Fuller	62
Berglunds snabbköp	Leverling	43
Blondel père et fils	Buchanan	80
Blondel père et fils	Fuller	50
Blondel père et fils	Leverling	99

....

And now to make a contingency table of Customers with Employees and Quantity we have:

```
SELECT Customers.CustomerName, Employees.LastName As EmployeeName,
ifnull(Sum(OrderDetails.Quantity),0) AS SumQuantity
FROM Customers
    CROSS JOIN Employees
    LEFT OUTER JOIN Orders ON Orders.EmployeeID=Employees.EmployeeID AND
Orders.CustomerID=Customers.CustomerID
    LEFT OUTER JOIN OrderDetails ON OrderDetails.OrderID=Orders.OrderID
GROUP BY Customers.CustomerName, Employees.LastName
ORDER BY Customers.CustomerName
```

Or (it's the same):

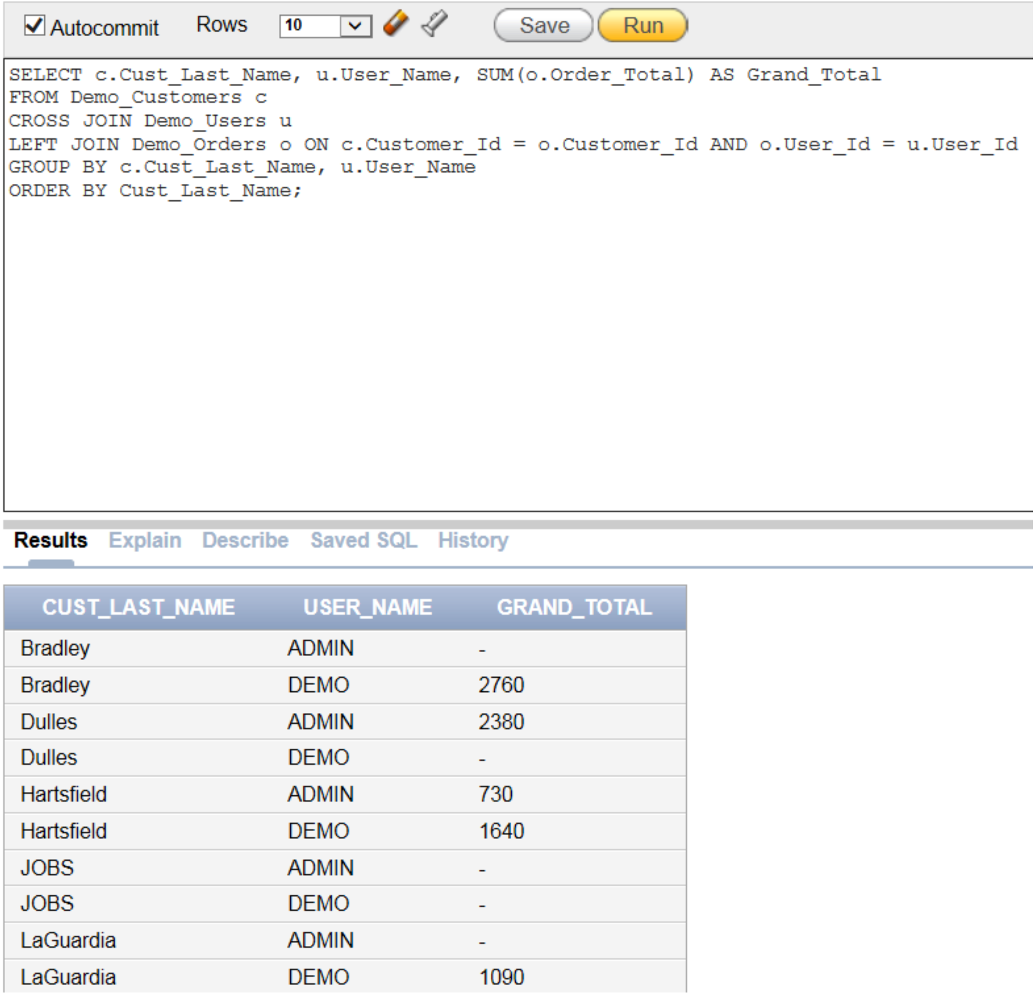
```
SELECT Customers.CustomerName,  Employees.LastName As EmployeeName,
ifnull(Sum(OrderDetails.Quantity),0) AS SumQuantity
FROM Customers
    CROSS JOIN Employees
    LEFT JOIN Orders ON Orders.EmployeeID=Employees.EmployeeID AND
Orders.CustomerID=Customers.CustomerID
    LEFT JOIN OrderDetails ON OrderDetails.OrderID=Orders.OrderID
GROUP BY Customers.CustomerName, Employees.LastName
ORDER BY Customers.CustomerName;
```

This will give:

CustomerName	EmployeeName	SumQuantity
Alfreds Futterkiste	Buchanan	0
Alfreds Futterkiste	Callahan	0
Alfreds Futterkiste	Davolio	0
Alfreds Futterkiste	Dodsworth	0
Alfreds Futterkiste	Fuller	0
Alfreds Futterkiste	King	0
Alfreds Futterkiste	Leverling	0
Alfreds Futterkiste	Peacock	0
Alfreds Futterkiste	Suyama	0
Alfreds Futterkiste	West	0
Ana Trujillo Emparedados y helados	Buchanan	0
Ana Trujillo Emparedados y helados	Callahan	0
Ana Trujillo Emparedados y helados	Davolio	0
Ana Trujillo Emparedados y helados	Dodsworth	0
Ana Trujillo Emparedados y helados	Fuller	0
Ana Trujillo Emparedados y helados	King	6
Ana Trujillo Emparedados y helados	Leverling	0
Ana Trujillo Emparedados y helados	Peacock	0
Ana Trujillo Emparedados y helados	Suyama	0
Ana Trujillo Emparedados y helados	West	0
Antonio Moreno Taquería	Buchanan	0
Antonio Moreno Taquería	Callahan	0
Antonio Moreno Taquería	Davolio	0
Antonio Moreno Taquería	Dodsworth	0
Antonio Moreno Taquería	Fuller	0
Antonio Moreno Taquería	King	0
Antonio Moreno Taquería	Leverling	24
Antonio Moreno Taquería	Peacock	0
Antonio Moreno Taquería	Suyama	0

Antonio Moreno Taquería	West	0
Around the Horn	Buchanan	0
Around the Horn	Callahan	55
Around the Horn	Davolio	0
Around the Horn	Dodsworth	0
Around the Horn	Fuller	0
Around the Horn	King	0
Around the Horn	Leverling	0
Around the Horn	Peacock	0
Around the Horn	Suyama	50
Around the Horn	West	0
B's Beverages	Buchanan	0
B's Beverages	Callahan	0
B's Beverages	Davolio	0
B's Beverages	Dodsworth	0
B's Beverages	Fuller	0
B's Beverages	King	39
B's Beverages	Leverling	0
B's Beverages	Peacock	0
B's Beverages	Suyama	0
B's Beverages	West	0
Berglunds snabbköp	Buchanan	0
Berglunds snabbköp	Callahan	64
Berglunds snabbköp	Davolio	0
Berglunds snabbköp	Dodsworth	0
Berglunds snabbköp	Fuller	62

or the equivalent in Oracle will give:




```sql
SELECT c.Cust_Last_Name, u.User_Name, SUM(o.Order_Total) AS Grand_Total
FROM Demo_Customers c
CROSS JOIN Demo_Users u
LEFT JOIN Demo_Orders o ON c.Customer_Id = o.Customer_Id AND o.User_Id = u.User_Id
GROUP BY c.Cust_Last_Name, u.User_Name
ORDER BY Cust_Last_Name;
```

CUST_LAST_NAME	USER_NAME	GRAND_TOTAL
Bradley	ADMIN	-
Bradley	DEMO	2760
Dulles	ADMIN	2380
Dulles	DEMO	-
Hartsfield	ADMIN	730
Hartsfield	DEMO	1640
JOBS	ADMIN	-
JOBS	DEMO	-
LaGuardia	ADMIN	-
LaGuardia	DEMO	1090

4.18.7 Exercise about a mixture of various joins in only one query

Here the purpose will be to understand and reproduce the following query:

Autocommit Rows 10 Save Run

```
SELECT Customers.ContactName, Orders.OrderID, sum(Order_Details.Quantity) AS Total, DT.GrandTotal AS GrandTotal,
   Employees.LastName AS Employee
FROM Customers
   FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
   LEFT OUTER JOIN Order_Details ON Orders.OrderID=Order_Details.ORDERID
   LEFT OUTER JOIN (SELECT Customers.ContactName, sum(Order_Details.Quantity) AS GrandTotal FROM Customers
      FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
      LEFT OUTER JOIN Order_Details ON Orders.OrderID=Order_Details.ORDERID
      GROUP BY Customers.ContactName
      ORDER BY Customers.ContactName) DT ON DT.ContactName=Customers.ContactName
   RIGHT OUTER JOIN Employees ON Employees.EmployeeID=Orders.EmployeeID
   GROUP BY Customers.ContactName, Orders.OrderID, DT.GrandTotal, Employees.LastName
   ORDER BY Customers.ContactName;
```

Results Explain Describe Saved SQL History

CONTACTNAME	ORDERID	TOTAL	GRANDTOTAL	EMPLOYEE
Alejandra Camino	10281	11	91	Peacock
Alejandra Camino	10282	8	91	Peacock
Alejandra Camino	10306	25	91	Davolio
Alejandra Camino	10917	11	91	Peacock
Alejandra Camino	11013	36	91	Fuller
Alexander Feuer	10277	32	172	Fuller
Alexander Feuer	10575	58	172	Buchanan
Alexander Feuer	10699	12	172	Leverling
Alexander Feuer	10779	40	172	Leverling
Alexander Feuer	10945	30	172	Peacock
More than 10 rows available. Increase rows selector to view more rows.				

10 rows returned in 0.13 seconds Download

4.18.8 SQL INTERSECT syntax

The SQL INTERSECT query allows you to return the results of 2 or more "select" queries. However, it only returns the rows selected by all queries. If a record exists in one query and not in the other, it will be omitted from the INTERSECT results.

Each SQL statement within the SQL INTERSECT query must have the same number of fields in the result sets with similar data types.

The syntax for the SQL INTERSECT query is:

```
select field1, field2, . field_n
from tables
INTERSECT
select field1, field2, . field_n
from tables;
```

As an example, we have using the W3School website the possibility to obtain all customers ID that have made an order:

```
SELECT CustomerId FROM Customers
INTERSECT
SELECT CustomerId
FROM Orders;
```

in other words: if a CustomId appears in both the Customers and Orders table, it would appear in your result set.

This is equivalent to an INNER JOIN with a GROUP but the INNER JOIN solution is more flexible because you can take the columns you want!:

```
SELECT Customers.CustomerID
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
GROUP BY Customers.CustomerID;
```

or more efficient with a DISTINCT (useful for MS Access):

```
SELECT DISTINCT Customers.CustomerID
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID=Orders.CustomerID;
```

4.18.9 SQL MINUS syntax

The SQL MINUS query returns all rows in the first SQL SELECT statement that are not returned in the second SQL SELECT statement.

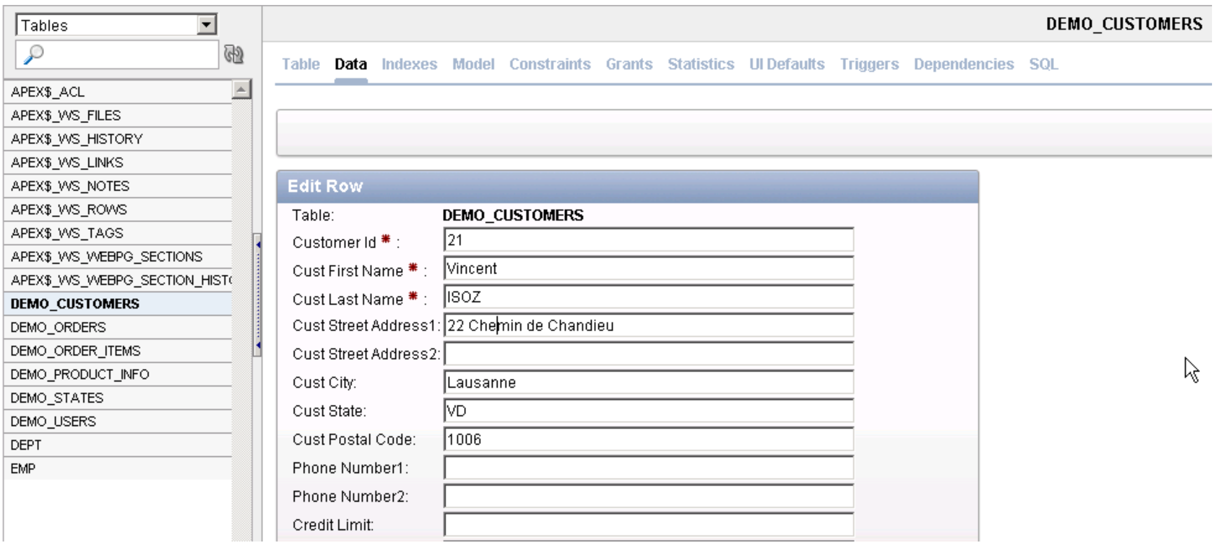
Each SQL SELECT statement within the SQL MINUS query must have the same number of fields in the result sets with similar data types.

The syntax for the SQL MINUS query is:

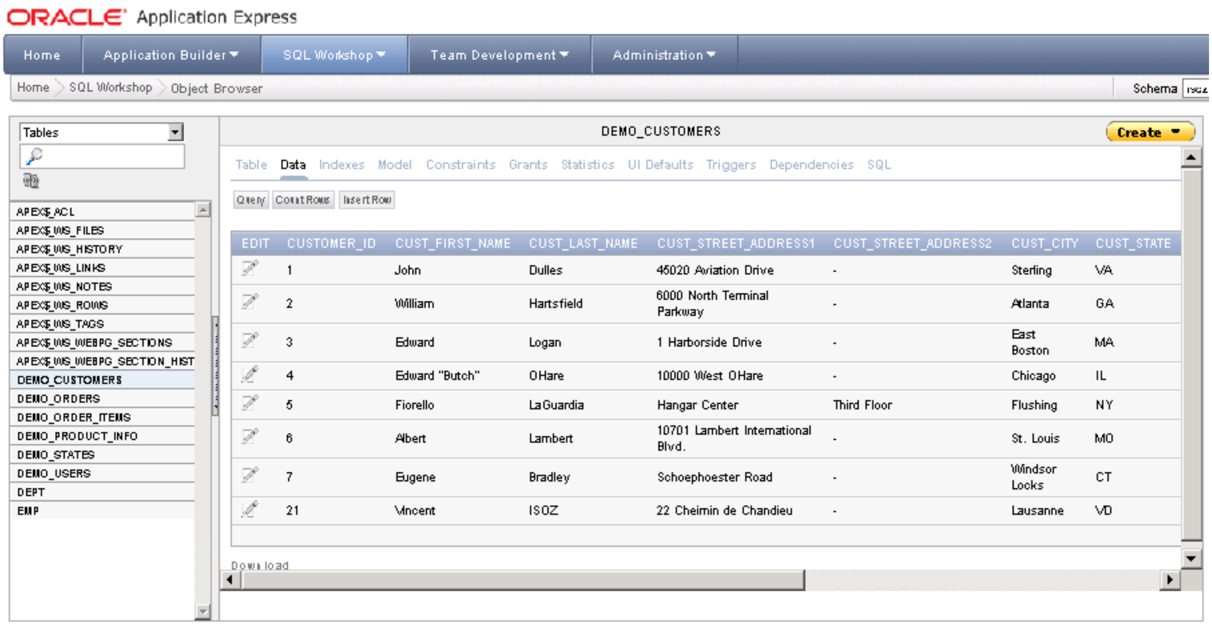
```
select field1, field2, ... field_n
from tables
MINUS
select field1, field2, ... field_n
from tables;
```

We can't use the MINUS statement on the W3School webiste. We will then focus with a small example on Oracle.

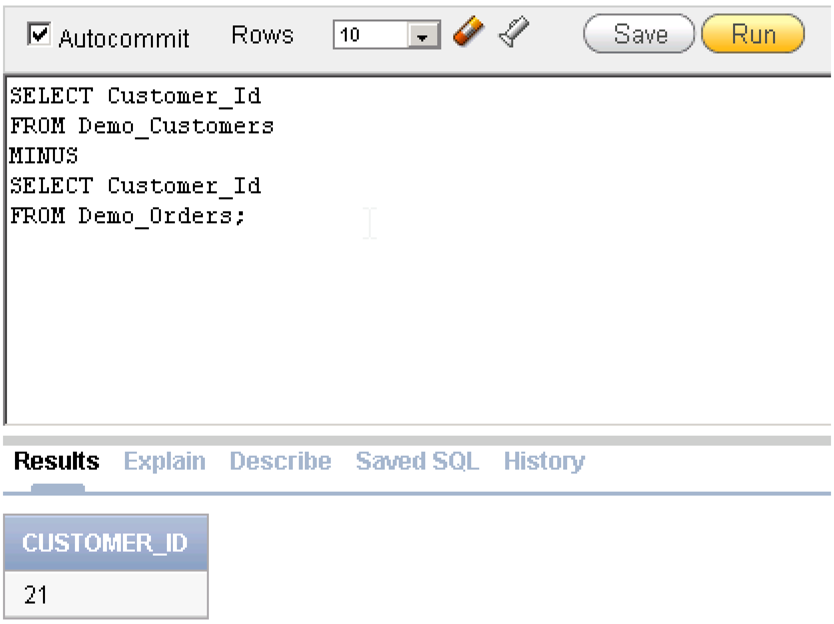
First in Oracle create a new customer in the customer table:



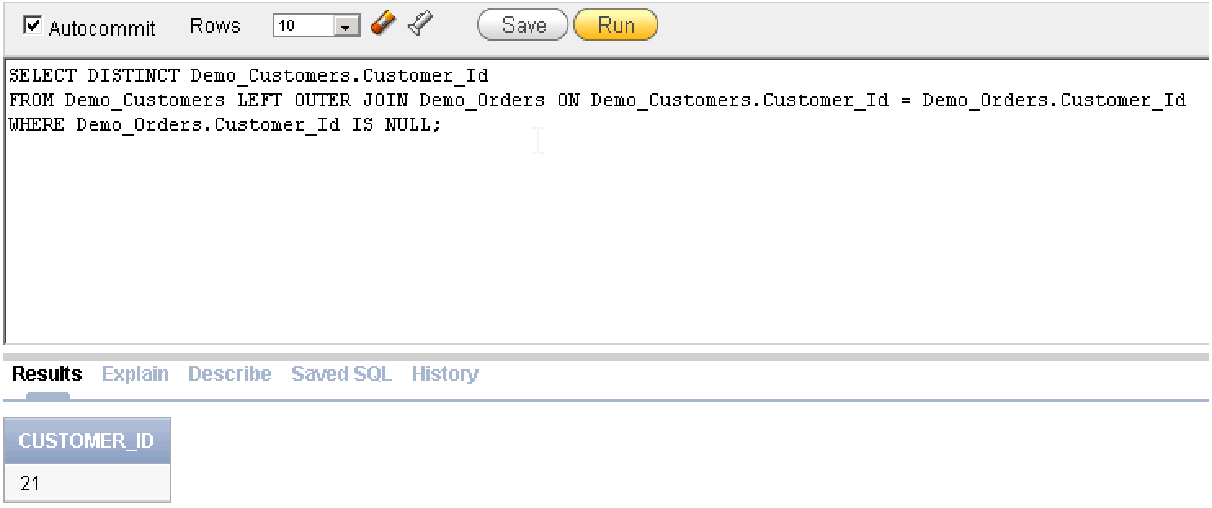
You will have then a new customer:



Now run the following query:



This can also be done with a LEFT OUTER JOIN (useful for MS Access for example) without the limitation of MINUS statement (possibility to take the columns you want):


Autocommit
Rows
10
Save
Run
SELECT DISTINCT Demo_Customers.Customer_Id
FROM Demo_Customers LEFT OUTER JOIN Demo_Orders ON Demo_Customers.Customer_Id = Demo_Orders.Customer_Id
WHERE Demo_Orders.Customer_Id IS NULL;
Results
Explain
Describe
Saved SQL
History
CUSTOMER_ID
21

4.19 SQL Nested Queries (Subqueries/Multiple Layers Queries)

A subquery is another powerful way of using SQL queries. One SQL statement can be embedded in another SQL statement.

A subquery is a SELECT statement within another SQL statement. The SQL statement can be SELECT, WHERE clause, FROM clause, JOIN, INSERT, UPDATE, DELETE, SET, DO, or another subquery.

The query that contains the subquery is normally called **outer query** and the subquery itself is **called inner** query.

If the subquery returns only one value, we speak about "**single value subquery**" or "**scalar subquery**".

If the subquery returns multiple values, we speak about "**row subquery**".

Advantages of using subquery

- Subqueries structure a complex query into isolated parts so that a complex query can be broken down into a series of logical steps for easy understanding and code maintenance.
- Subqueries allow you to use the results of another query in the outer query.
- In some cases, subqueries can replace complex joins and unions and subqueries are easier to understand.

Disadvantages of using subquery

When subquery is used, the database server (actually the query optimizer) may need to perform additional steps, such as sorting, before the results from the subquery are used. If a query that contains subqueries can be rewritten as a join, you should use join rather than subqueries. This is because using join typically allows the query optimizer to retrieve data in the most efficient way. In other words, the optimizer is more mature for MySQL for joins than for subqueries, so in many cases a statement that uses a subquery can be executed more efficiently if you rewrite it as a join.

Rules that govern the use of subqueries

- A subquery must always appear within parentheses.
- You can embed a subquery inside another one. You can have as many level as you need.
- If the outer query expects a single value or a list of values from the subquery, the subquery can only use one expression or column name in its select list.
- When you use the result from a subquery to join a table in a JOIN operation, no index can be used on the join column(s). This is because subquery first generates result on the fly and then the result is used in the join.

4.19.1 Scalar subqueries (single-value subquery) examples

When the subquery returns a single value, the subquery is only evaluated once and then the value is returned to outer query to use. This kind of subqueries are also known as single-value subquery or scalar subquery.

This query returns data for all customers and their orders where the orders were shipped on the most recent recorded day.

```
SELECT OrderID, CustomerID
FROM Orders
WHERE OrderDate=(SELECT MAX(OrderDate) FROM ORDERS);
```

This query returns all products whose unit price is greater than average unit price.

```
SELECT DISTINCT ProductName, Price
FROME Products
WHERE Price>(SELECT AVG(UnitPrice) FROM Products)
ORDER BY UnitPrice DESC;
```

4.19.2 Column subqueries (multiple values query using one column) examples

When the subquery returns a list of values, the subquery is only evaluated once and then the list of values is returned to outer query to use. This kind of subqueries are also known as "column subquery".

This query retrieves a list of customers that made purchases after the date 1997-02-05.

```
SELECT CustomerName, Country
FROM Customers
WHERE CustomerID in
(
   SELECT CustomerID
   FROM Orders
   WHERE OrderDate > '1997-02-05'
);
```

The query below returns the same result (on the W3 School website!) as query above because the list of CustomerIDs are used rather than the subquery:

```
SELECT CustomerID, Country
FROM Customers
WHERE CustomerID in
(
    'Ernst Handel',
    'Mère Paillarde',
    'Old World Delicatessen',
    'Reggiani Caseifici',
    'Save-a-lot Markets',
    'Toms Spezialitäten'
);
```

For sure the same result can be obtained using the JOIN statement (often, a query that contains subqueries can be rewritten as a join). Using inner join allows the query optimizer to retrieve data in the most efficient way:

```
SELECT a.CustomerID, a.Country
FROM Customers AS a
INNER JOIN Orders AS b ON a.CustomerID = b.CustomerID
WHERE b.OrderDate > '1997-02-05'
```

4.19.3 Row subqueries (multiple values query using multiple column) examples

The below statement semi-joins Customers to Suppliers based on a tuple comparison, not just a single column comparison. This is useful, for example, when you want to select all Customers from a table whose City AND Country are also contained in the Suppliers table (without any formal foreign key relationship, of course):

```
SELECT *
FROM Customers
WHERE (City,Country) IN (
    SELECT City, Country FROM Suppliers
)
```

This example won't work on the W3 School website due to implementation limitation of the web interface (see the alternative below with the green background). We also won't lose time to import a database to test this in Oracle.

Some systems want's the following syntax:

```
SELECT *
FROM Customers
WHERE ROW(City,Country) IN (
    SELECT City, Country FROM Suppliers
)
```

If none of the above works you cans use the EXIST statement (see later) with the following syntax (this will work on the W3 School website):

```
SELECT *
FROM Customers
WHERE  EXISTS (
    SELECT * FROM Suppliers
    WHERE Customers.City= Suppliers.City AND Customers.Country =
Suppliers.Country
)
```

4.19.4 Correlated subqueries examples

The name of correlated subqueries means that a subquery is correlated with the outer query. The correlation comes from the fact that the subquery uses information from the outer query and the subquery executes once for every row in the outer query.

A correlated subquery can usually be rewritten as a join query. Using joins enables the database engine to use the most efficient execution plan. The query optimizer is more mature for joins than for subqueries, so in many cases a statement that uses a subquery should normally be rephrased as a join to gain the extra speed in performance.

Note that alias must be used to distinguish table names in the SQL query that contains correlated subqueries.

For example, the previous query:

```
SELECT *
FROM Customers
WHERE EXISTS (
    SELECT * FROM Suppliers
    WHERE Customers.City= Suppliers.City AND Customers.Country =
Suppliers.Country
)
```

belongs to the family of correlated subqueries because the subquery use the Customers.City and Customers.Country attributes of the outer query.

The query below query finds out a list of orders and their customers who ordered more than 20 items of ProductID 6 on a single order.

```
SELECT a.OrderID,
       a.CustomerID
FROM Orders AS a
WHERE
(
    SELECT Quantity
    FROM OrderDetails as b
    WHERE a.OrderID = b.OrderID and b.ProductID = 6
) > 20;
```

4.19.5 SQL EXIST function

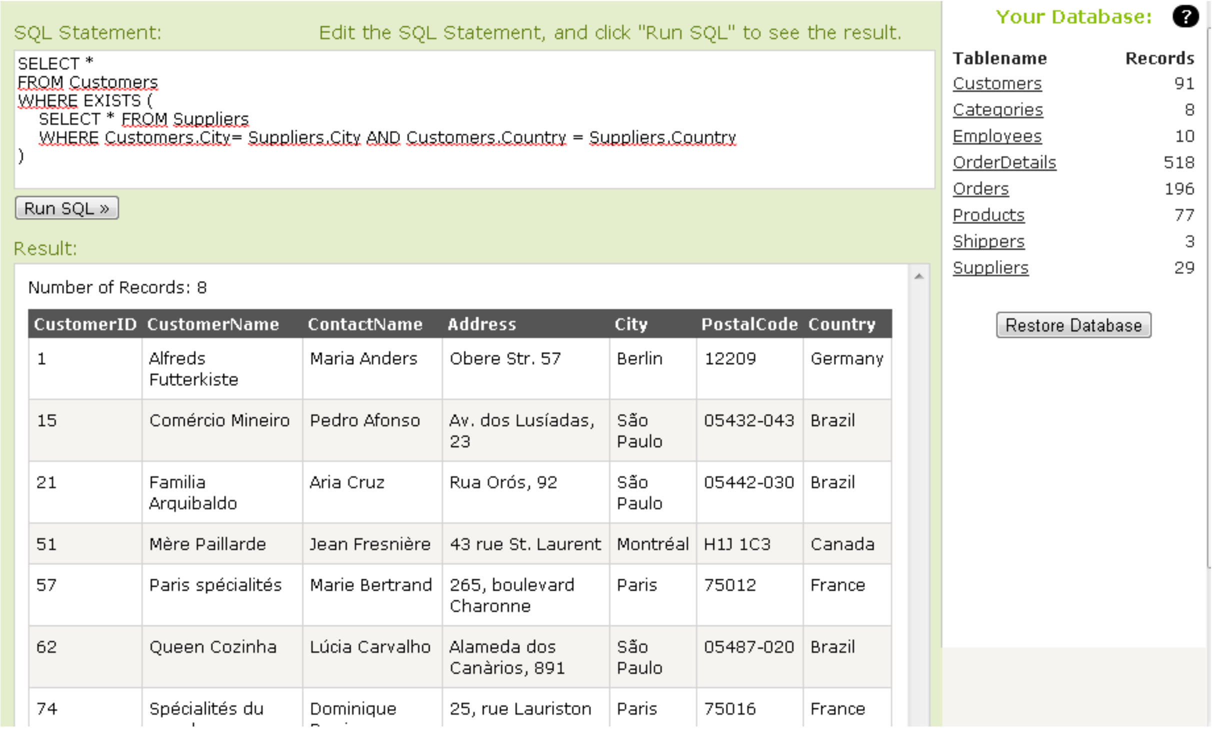
EXISTS is used with a correlated subquery in WHERE clause to examine if the result the subquery returns is TRUE or FALSE. The true or false value is then used to restrict the rows from outer query select. Because EXISTS only return TRUE or FALSE in the subquery, the SELECT list in the subquery does not need to contain actual column name(s). Normally use SELECT * (asterisk) is sufficient but you can use SELECT column1, column2, ... or anything else. It does not make any difference.

Because EXISTS are used with correlated subqueries, the subquery executes once for every row in the outer query. In other words, for each row in outer query, by using information from the outer query, the subquery checks if it returns TRUE or FALSE, and then the value is returned to outer query to use.

Remember we already saw such an example (all Customers that have a Supplier in the same City and Country as their home address):

```
SELECT *
FROM Customers
WHERE EXISTS (
    SELECT * FROM Suppliers
    WHERE Customers.City= Suppliers.City AND Customers.Country =
Suppliers.Country
)
```

that returns 8 rows on the 91 customers:



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
15	Comércio Mineiro	Pedro Afonso	Av. dos Lusíadas, 23	São Paulo	05432-043	Brazil
21	Familia Arquibaldo	Aria Cruz	Rua Orós, 92	São Paulo	05442-030	Brazil
51	Mère Paillarde	Jean Fresnière	43 rue St. Laurent	Montréal	H1J 1C3	Canada
57	Paris spécialités	Marie Bertrand	265, boulevard Charonne	Paris	75012	France
62	Queen Cozinha	Lúcia Carvalho	Alameda dos Canàrios, 891	São Paulo	05487-020	Brazil
74	Spécialités du	Dominique	25, rue Lauriston	Paris	75016	France

Tablename	Records
Customers	91
Categories	8
Employees	10
OrderDetails	518
Orders	196
Products	77
Shippers	3
Suppliers	29

But don't forget that this can also be done with a JOIN statement!

4.19.6 SQL NOT EXISTS function

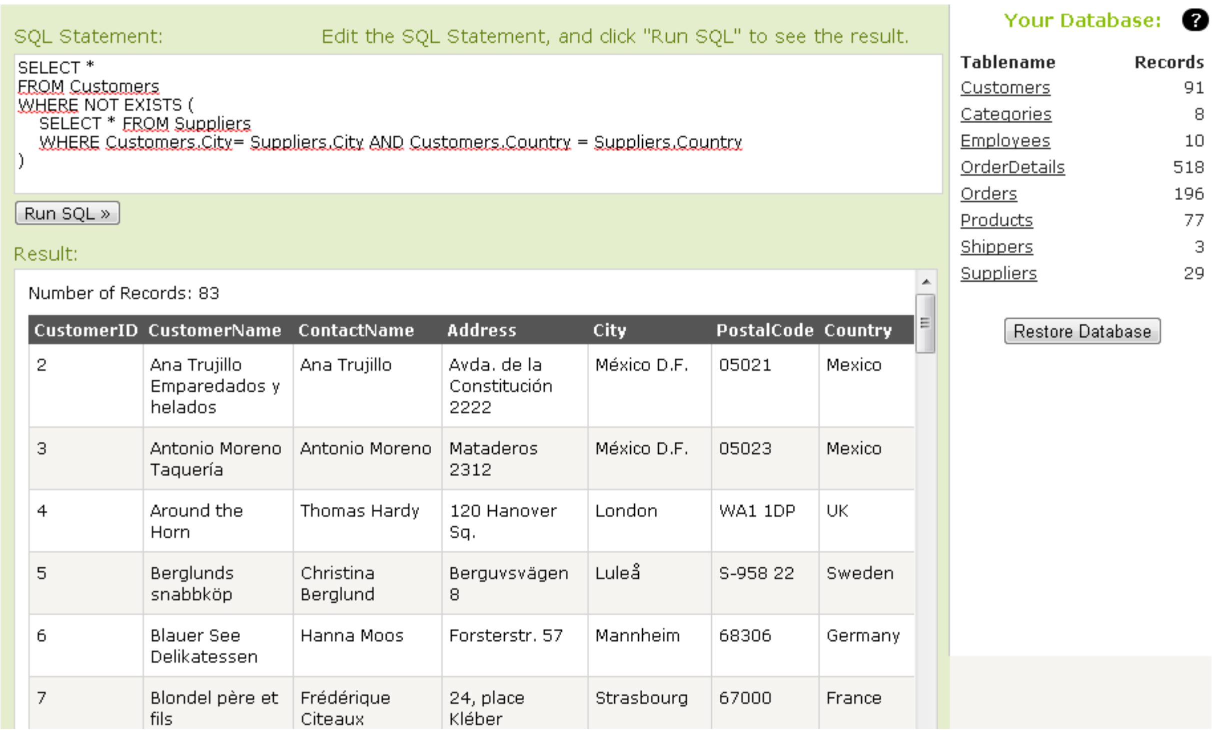
NO EXISTS is used with a correlated subquery in WHERE clause to examine if the result the subquery returns is TRUE or FALSE. The true or false value is then used to restrict the rows from outer query select. Because NO EXISTS only return TRUE or FALSE in the subquery, the SELECT list in the subquery does not need to contain actual column name(s). Normally use SELECT * (asterisk) is sufficient but you can use SELECT column1, column2, ... or anything else. It does not make any difference.

Because NO EXISTS are used with correlated subqueries, the subquery executes once for every row in the outer query. In other words, for each row in outer query, by using information from the outer query, the subquery checks if it returns TRUE or FALSE, and then the value is returned to outer query to use.

We will take as example the opposite of the previous example:

```
SELECT *
FROM Customers
WHERE NOT EXISTS (
    SELECT * FROM Suppliers
    WHERE Customers.City= Suppliers.City AND Customers.Country =
Suppliers.Country
)
```

that returns 91-8=83 rows:


SQL Statement: Edit the SQL Statement, and click "Run SQL" to see the result.

```
SELECT *
FROM Customers
WHERE NOT EXISTS (
    SELECT * FROM Suppliers
    WHERE Customers.City= Suppliers.City AND Customers.Country = Suppliers.Country
)
```

Run SQL »

Result:

Number of Records: 83

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France

Your Database:

Tablename	Records
Customers	91
Categories	8
Employees	10
OrderDetails	518
Orders	196
Products	77
Shippers	3
Suppliers	29

Restore Database

4.19.7 ALL, ANY and SOME

It is quite possible you could work with Oracle databases for many years and never come across the ALL, ANY and SOME comparison conditions in SQL because there are alternatives to them that are used more regularly. If you are planning to sit the Oracle Database SQL Expert (1Z0-047) exam you should be familiar with these conditions as they are used frequently in the questions.

For the examples below we will use the following EMP Oracle table:

EDIT	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
	7839	KING	PRESIDENT	-	11/17/1981	5000	-	10
	7698	BLAKE	MANAGER	7839	05/01/1981	2850	-	30
	7782	CLARK	MANAGER	7839	06/09/1981	2450	-	10
	7566	JONES	MANAGER	7839	04/02/1981	2975	-	20
	7788	SCOTT	ANALYST	7566	12/09/1982	3000	-	20
	7902	FORD	ANALYST	7566	12/03/1981	3000	-	20
	7369	SMITH	CLERK	7902	12/17/1980	800	-	20
	7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
	7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
	7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
	7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
	7876	ADAMS	CLERK	7788	01/12/1983	1100	-	20
	7900	JAMES	CLERK	7698	12/03/1981	950	-	30
	7934	MILLER	CLERK	7782	01/23/1982	1300	-	10

row(s) 1 - 14 of 14

4.19.7.1 ALL

The ALL comparison condition is used to compare a value to a list or subquery. It must be preceded by =, !=, >, <, <=, >= and followed by a list or subquery.

When the ALL condition is followed by a list, the optimizer expands the initial condition to all elements of the list and strings them together **with AND operators**, as shown below.

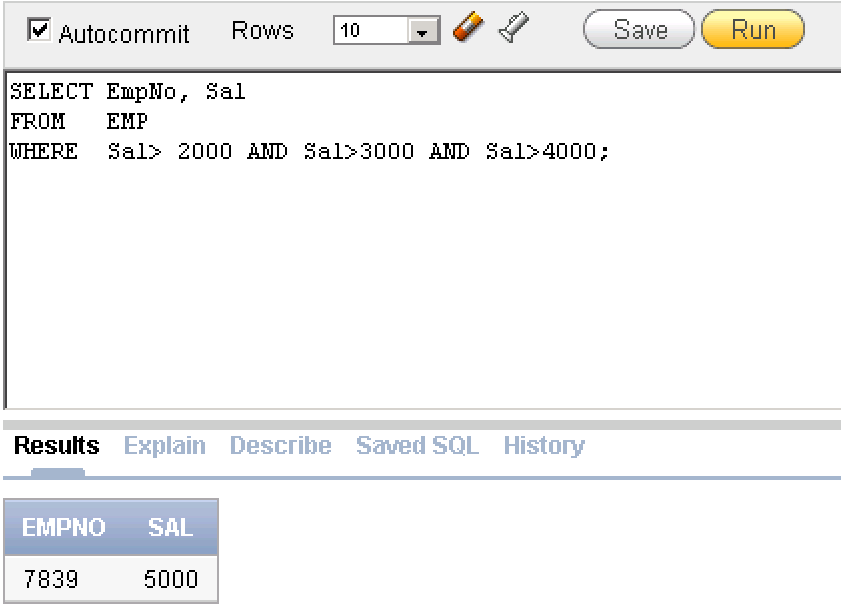
Autocommit Rows 10 Save Run

```
SELECT EmpNo, Sal
FROM   EMP
WHERE  SAL > ALL (2000, 3000, 4000);
```

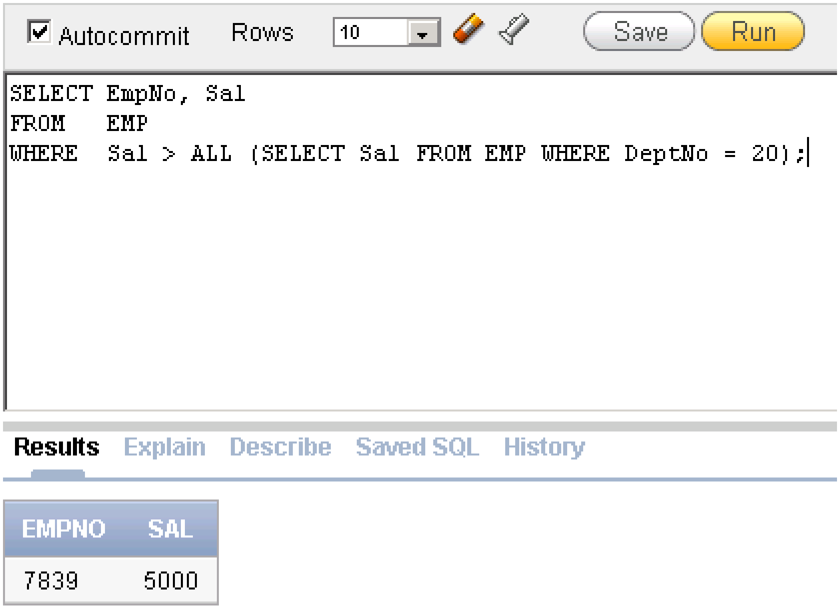
Results Explain Describe Saved SQL History

EMPNO	SAL
7839	5000

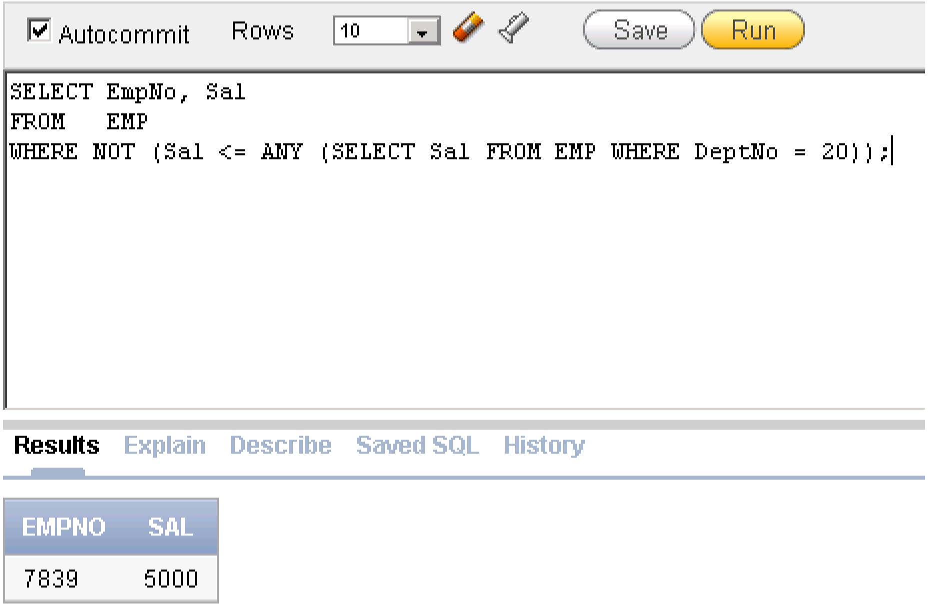
Transformed to equivalent statement without ALL:



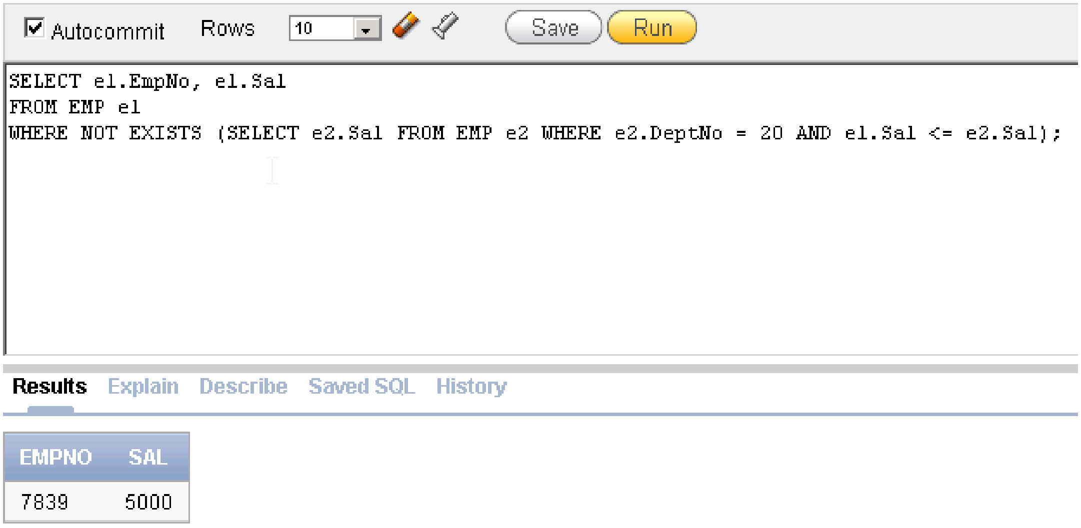
When the ALL condition is followed by a subquery, the optimizer performs a two-step transformation as shown below.



Transformed to equivalent statement using ANY (not really intuitive):



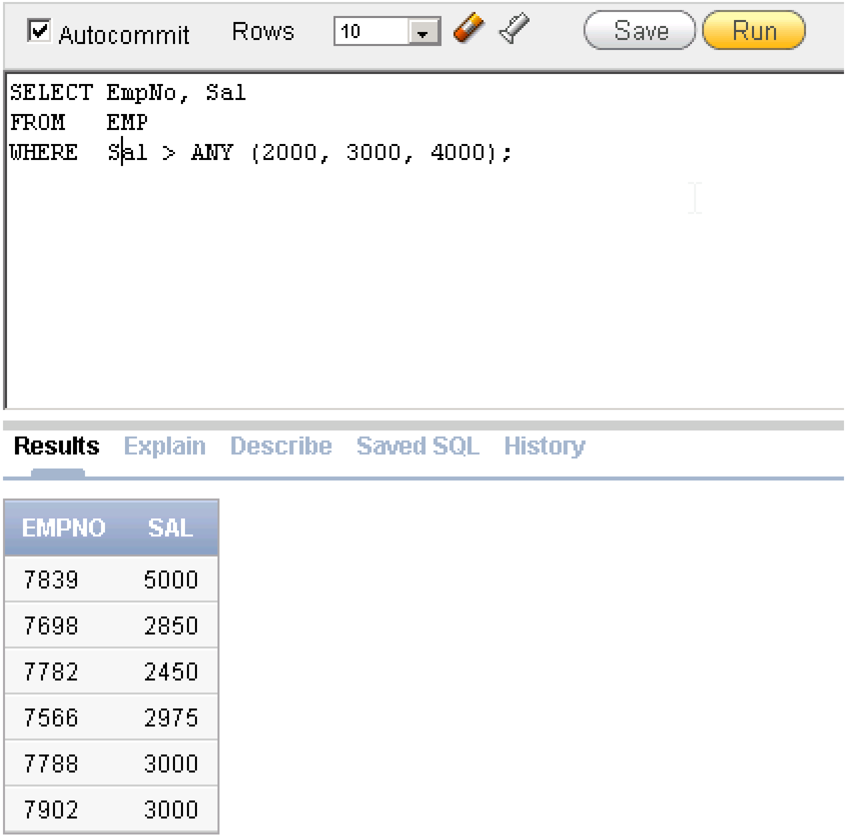
or transformed to equivalent statement without ANY (also not really intutive):



4.19.7.2 ANY

The ANY comparison condition is used to compare a value to a list or subquery. It must be preceded by =, !=, >, <, <=, >= and followed by a list or subquery.

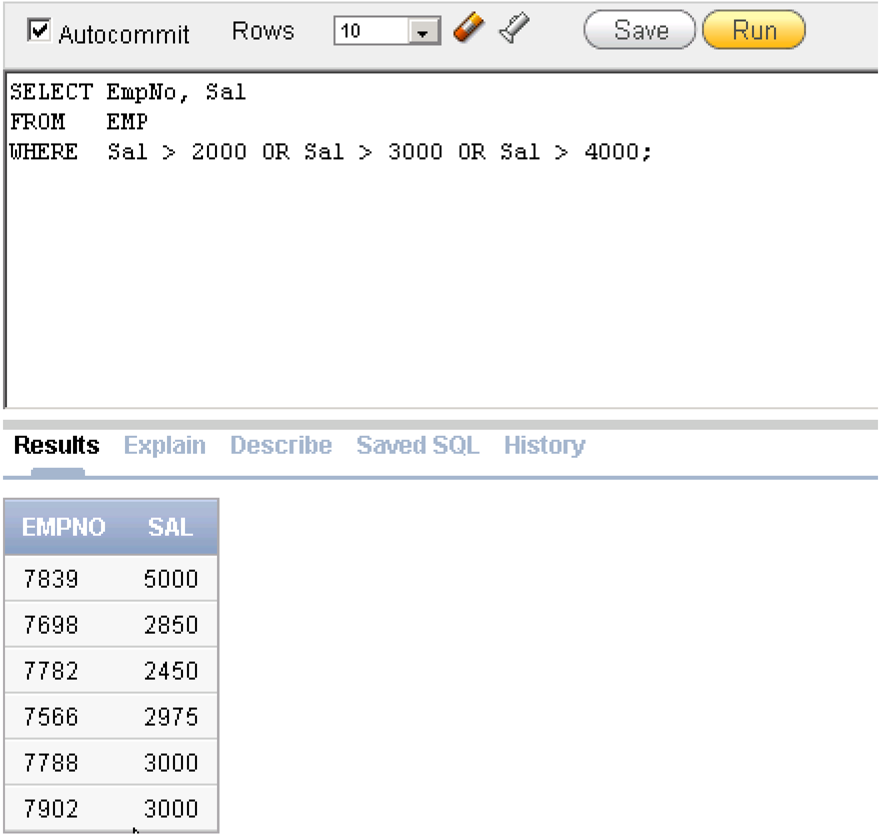
When the ANY condition is followed by a list, the optimizer expands the initial condition to all elements of the list and strings them together **with OR operators**, as shown below.




```sql
SELECT EmpNo, Sal
FROM   EMP
WHERE  Sal > ANY (2000, 3000, 4000);
```

EMPNO	SAL
7839	5000
7698	2850
7782	2450
7566	2975
7788	3000
7902	3000

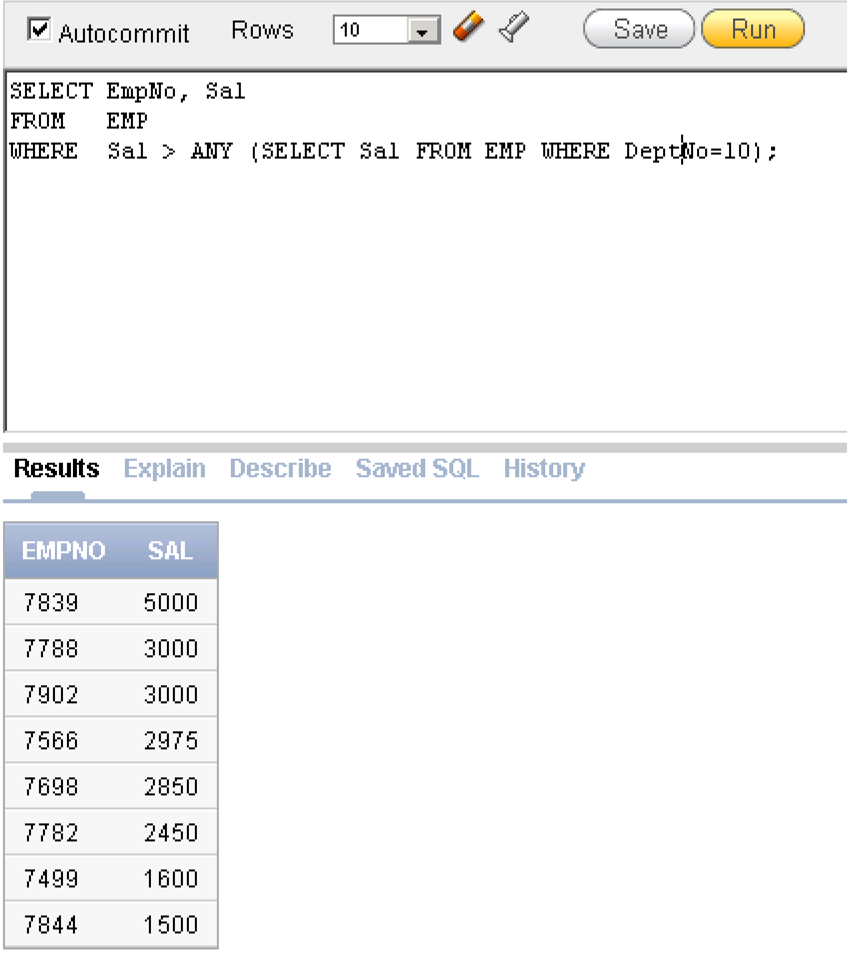
Transformed to equivalent statement without ANY:




```
SELECT EmpNo, Sal
FROM   EMP
WHERE  Sal > 2000 OR Sal > 3000 OR Sal > 4000;
```

EMPNO	SAL
7839	5000
7698	2850
7782	2450
7566	2975
7788	3000
7902	3000

When the ANY condition is followed by a subquery, the optimizer performs a single transformation as shown below:




```sql
SELECT EmpNo, Sal
FROM   EMP
WHERE  Sal > ANY (SELECT Sal FROM EMP WHERE DeptNo=10);
```

EMPNO	SAL
7839	5000
7788	3000
7902	3000
7566	2975
7698	2850
7782	2450
7499	1600
7844	1500

the transformed to equivalent statement without ANY:

```sql
SELECT EmpNo, Sal
FROM EMP e1
WHERE EXISTS (SELECT e2.sal FROM EMP e2 WHERE e2.deptno = 10 AND e1.sal > e2.sal);
```

EMPNO	SAL
7839	5000
7788	3000
7902	3000
7566	2975
7698	2850
7782	2450
7499	1600
7844	1500

4.19.7.3 SOME

The SOME and ANY comparison conditions do exactly the same thing and are completely interchangeable!

5 SQL for DDL (Data Definition Language)

SQL DML has to do with the "physicial" position/creation/deletion of datas in the database.

5.1 SQL SELECT INTO statement

With SQL, you can copy information from one table into another.

The SELECT INTO statement copies data from one table and inserts it into a **new table**.

Examples:

We can copy all columns into the new table:

```
SELECT *
INTO newtable [IN externaldb]
FROM table1;
```

Or we can copy only the columns we want into the new table:

```
SELECT column_name(s)
INTO newtable [IN externaldb]
FROM table1;
```

Tip: The new table will be created with the column-names and types as defined in the SELECT statement. You can apply new names using the AS clause.

The examples below won't work on W3 Schools website or even in Oracle (see the lasts queries in the screenshots to see how to do this in Oracle) or MySQL but will directly work with MS Access

Create a backup copy of Customers:

```
SELECT *
INTO CustomersBackup2013
FROM Customers;
```

Use the IN clause to copy the table into another database:

```
SELECT *
INTO CustomersBackup2013 IN 'Backup.mdb'
FROM Customers;
```

Copy only a few columns into the new table:

```
SELECT CustomerName, ContactName
INTO CustomersBackup2013
FROM Customers;
```

Copy only the German customers into the new table:

```
SELECT *
INTO CustomersBackup2013
FROM Customers
WHERE Country='Germany';
```

Copy data from more than one table into the new table:

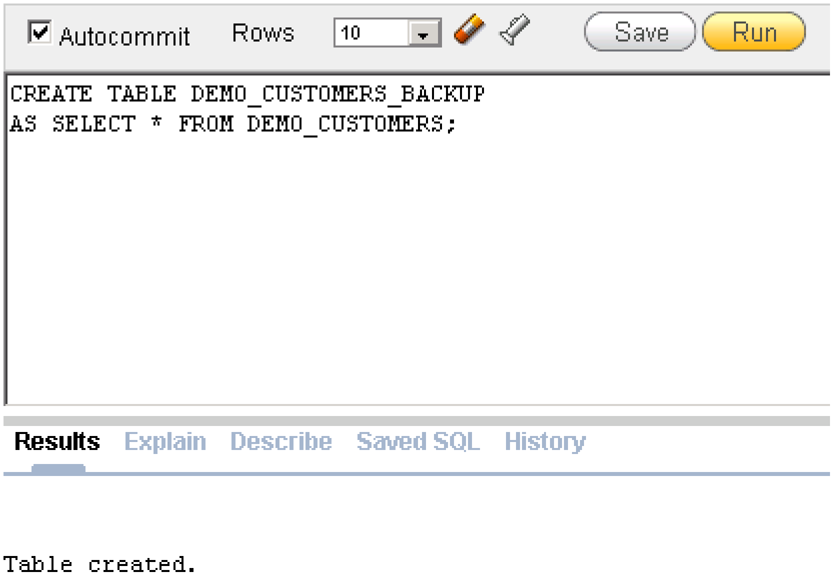
```
SELECT Customers.CustomerName, Orders.OrderID
INTO CustomersOrderBackup2013
FROM Customers
```

```
LEFT JOIN Orders
ON Customers.CustomerID=Orders.CustomerID;
```

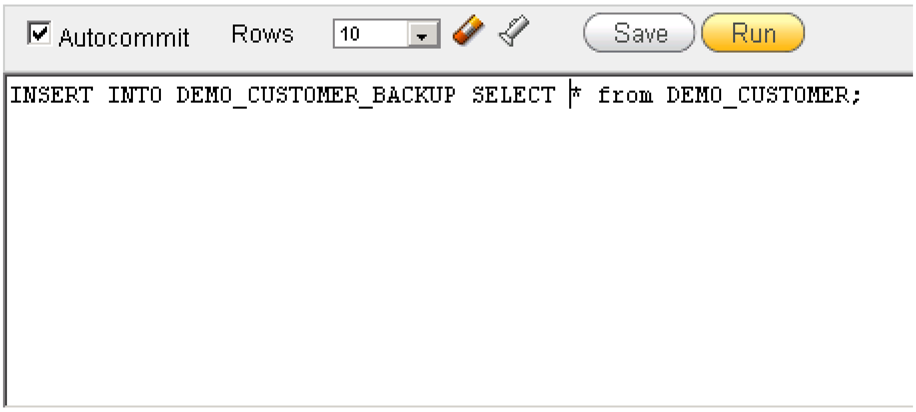
Tip: The SELECT INTO statement can also be used to create a new, empty table using the schema of another. Just add a WHERE clause that causes the query to return no data:

```
SELECT *
INTO newtable
FROM table1
WHERE 1=0;
```

In Oracle you will have to run if the table does not already exist:



and if the table already exists:



5.2 SQL INSERT SELECT INTO statement

The INSERT INTO SELECT statement selects data from one table and inserts it **into an existing table**. Any existing rows in the target table are unaffected.

We can copy all columns from one table to another, existing table:

```
INSERT INTO table2
SELECT * FROM table1;
```

Or we can copy only the columns we want to into another, existing table:

```
INSERT INTO table2
(column_name(s))
SELECT column_name(s)
FROM table1;
```

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

And a selection from the "Suppliers" table:

SupplierID	SupplierName	ContactName	Address	City	Postal Code	Country	Phone
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	Londona	EC1 4SD	UK	(171) 555-2222
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA	(100) 555-4822
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA	(313) 555-5735

Copy only a few columns from "Suppliers" into "Customers":

```
INSERT INTO Customers (CustomerName, Country)
SELECT SupplierName, Country FROM Suppliers;
```

Copy only the German suppliers into "Customers":

```
INSERT INTO Customers (CustomerName, Country)
SELECT SupplierName, Country FROM Suppliers
WHERE Country='Germany';
```

5.3 SQL CREATE DATABASE statement

The CREATE DATABASE statement is used to create a database.

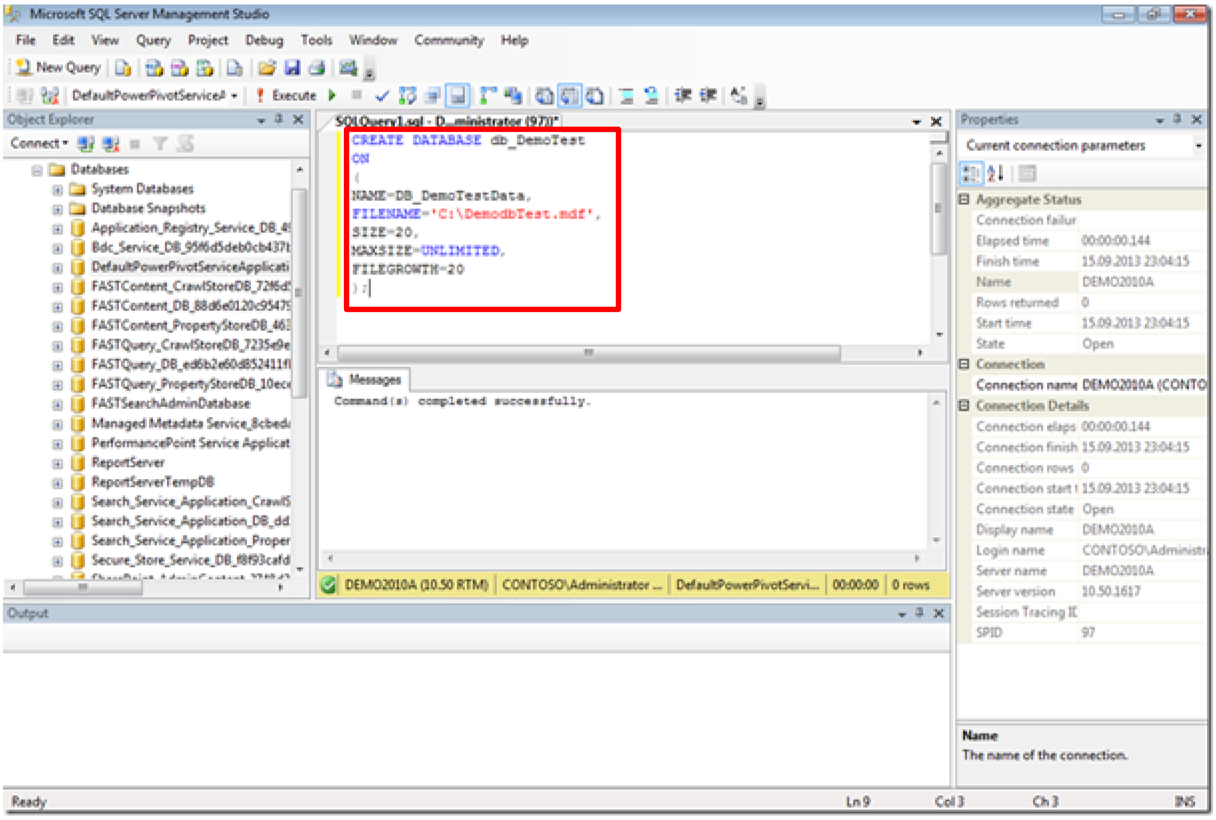
SQL CREATE DATABASE Syntax:

```
CREATE DATABASE dbname;
```

It will not be possible to create a database in Oracle Express because the first and only database is created during installation with the CREATE DATABASE Statement. In MS Access and on W3 School you can also not use CREATE DATABASE statement.

5.3.1 On SQL Server

You just open SQL Server (here you can see SQL Server 2008 R2) and you type the following query:

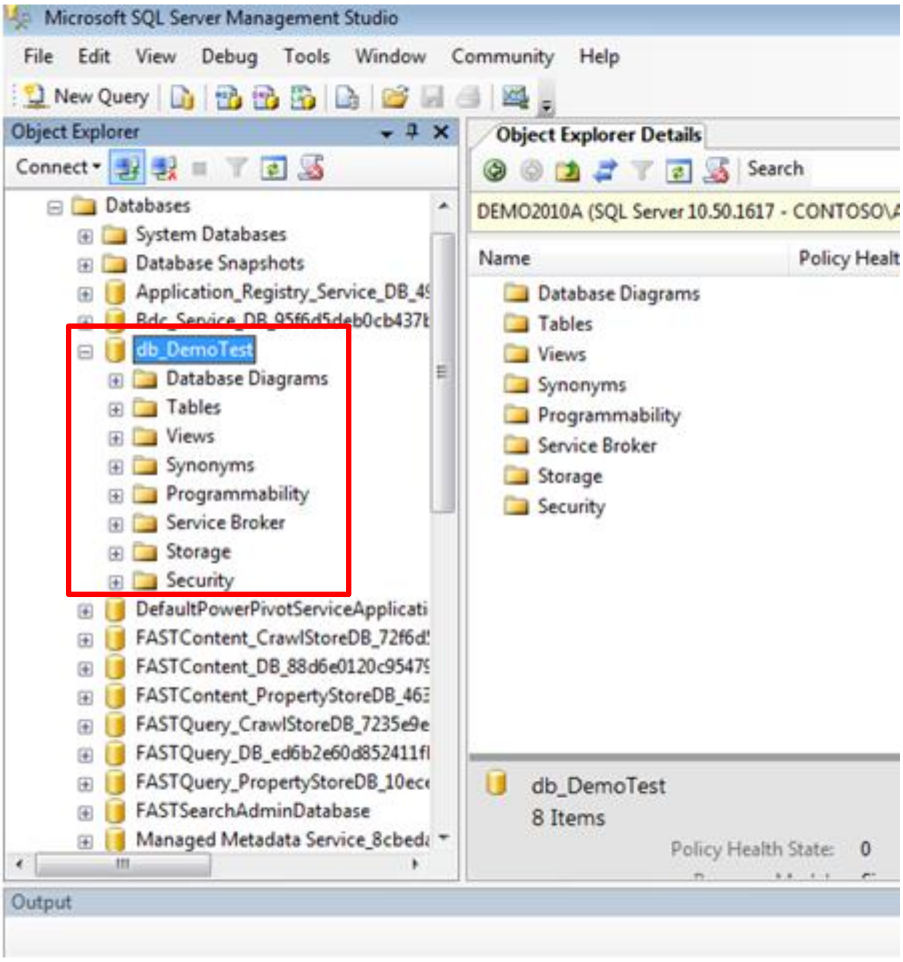


This example query create a database named as db_DemoTest in this case I omitted PRIMARY option and the first file is assumed as a primary file. The logical name of this file is DB_DemoTestData as I mentioned in query. File name parameter is for specify physical location for the database file *.mdf in Local disk C:\ in my hard drive.

The original size of this file is 20MB, Additional 20MB from disk may allocated by the system if it needed (FILEGROWTH).

If MAXSIZE option is not specified or it set to unlimited the file will dynamically use all space in disk as it grows.

You close an reopen Microsoft SQL Server Management Studio and then you will see:



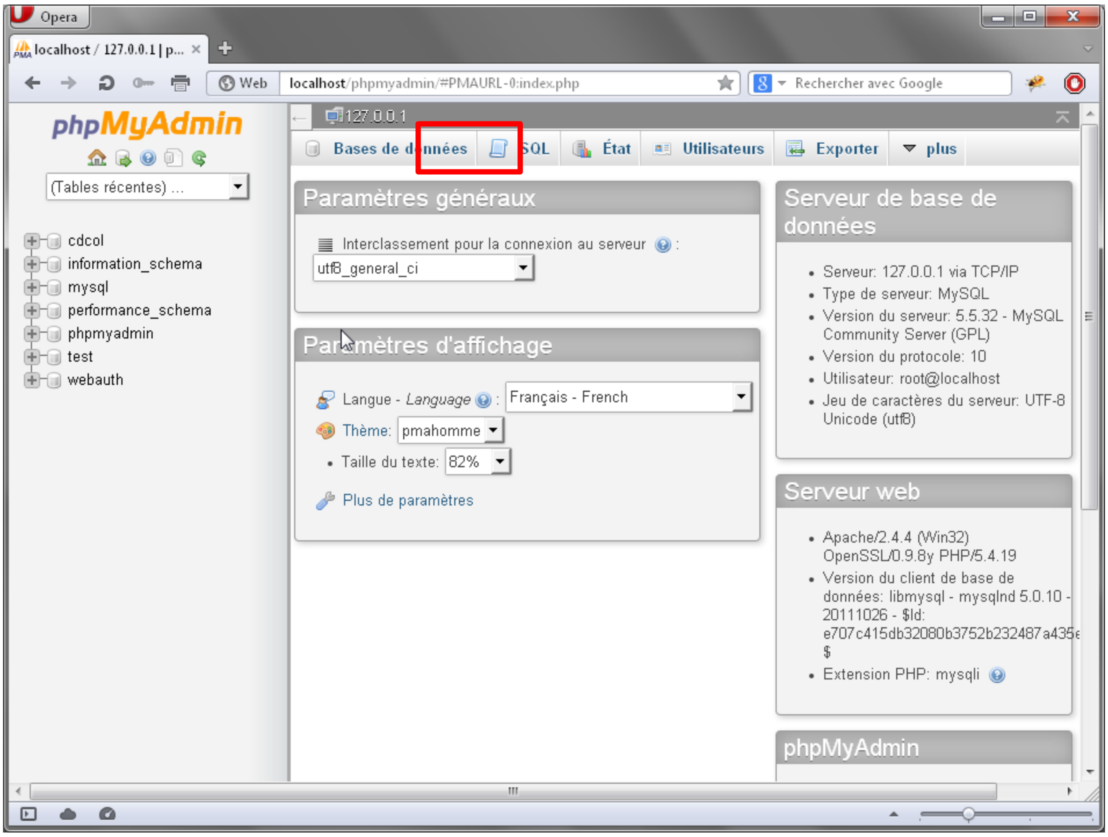
5.3.2 On mySQL

For the example we will download and install XAMP:

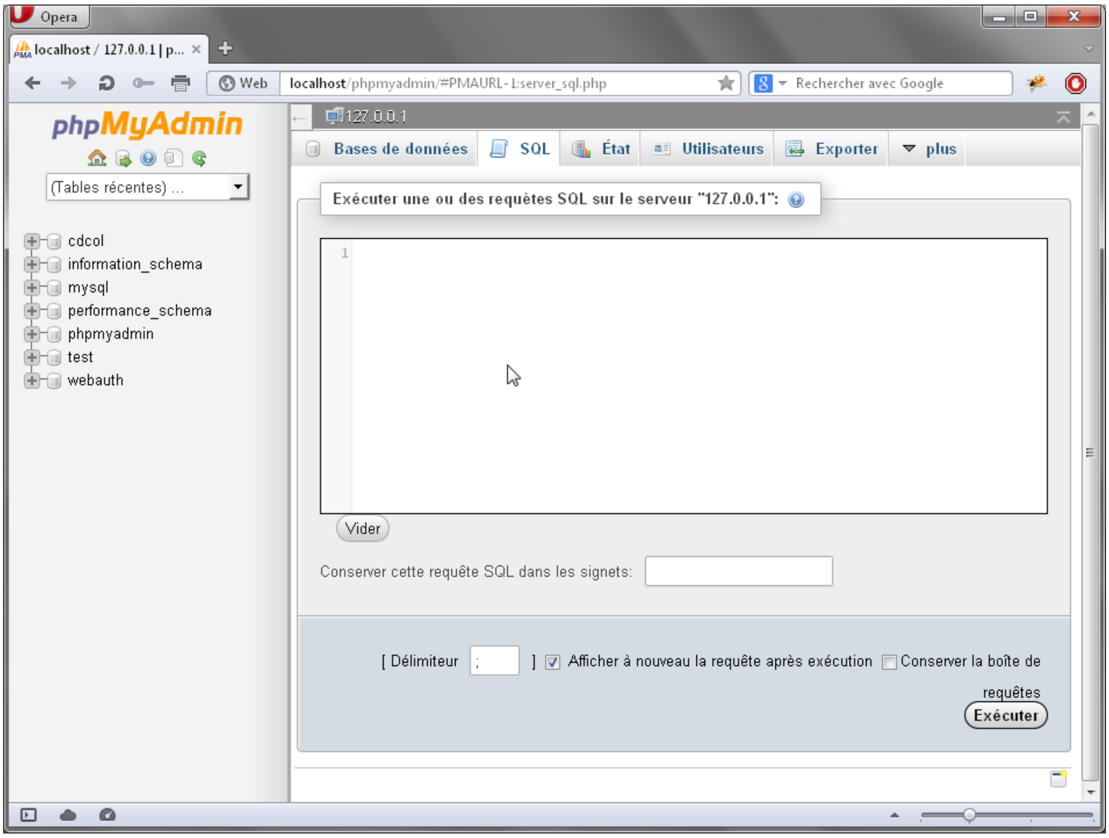
http://www.apachefriends.org/fr/xampp.html

After installation:

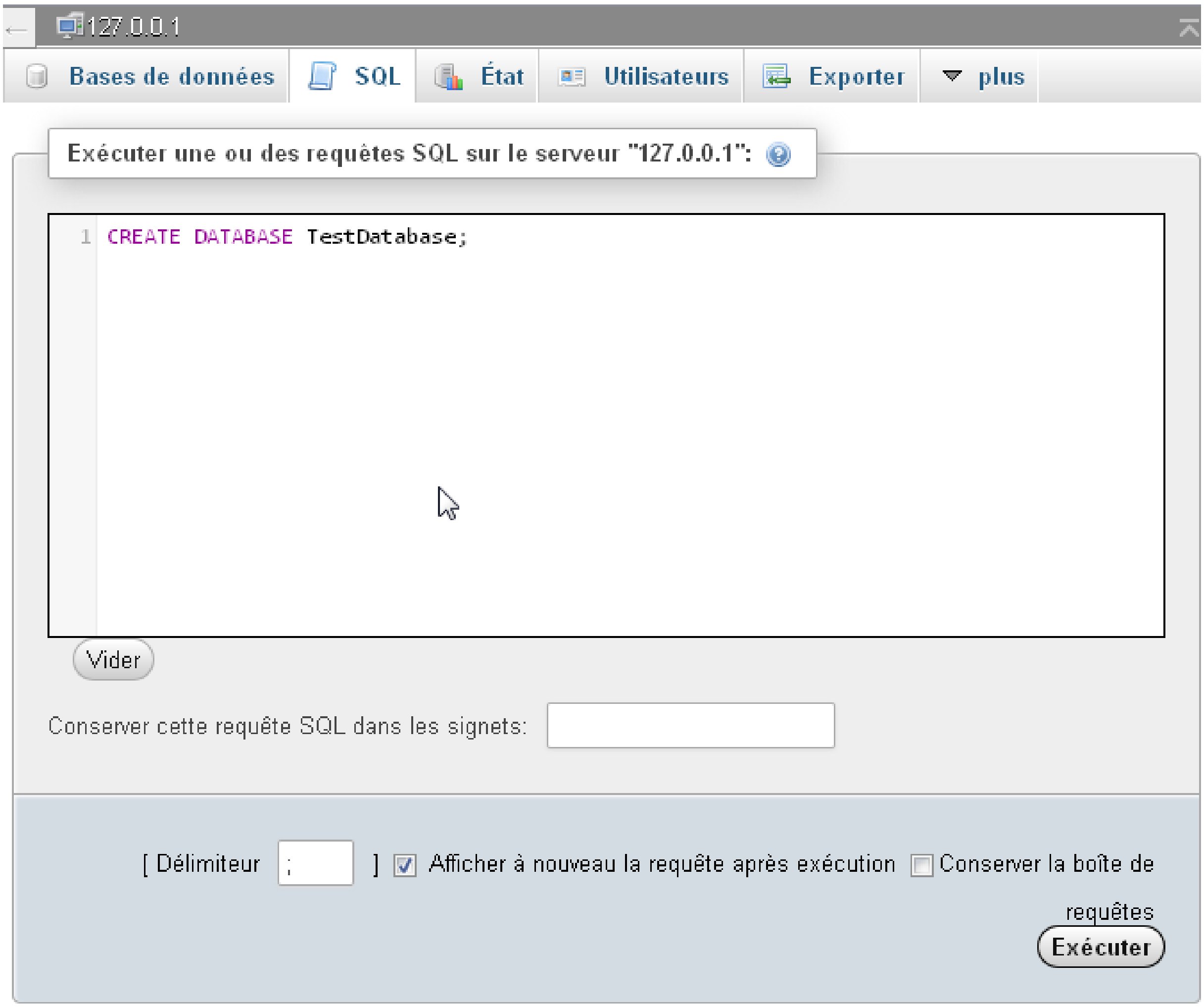
Clic on phpMyAdmin:



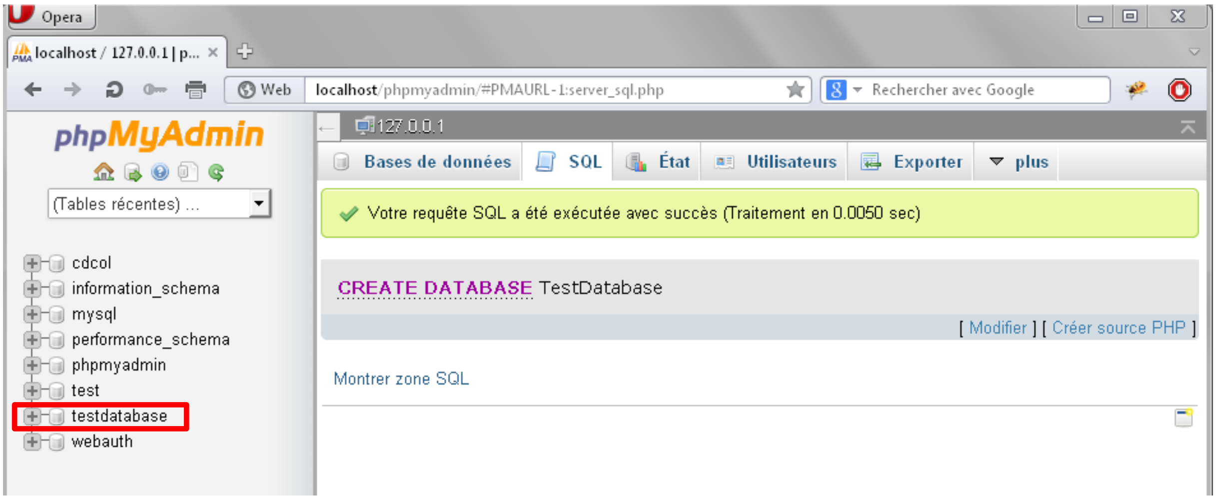
Clic on SQL:



Now type:



If you click on **Exécuter** you will have:



5.4 SQL CREATE TABLE statement

5.4.1 With Data Types statements only

The CREATE TABLE statement is used to create an empty table in a database.

Tables are organized into rows and columns; and each table must have a name.

SQL CREATE TABLE Syntax:

```
CREATE TABLE table_name
(
column_name1 data_type(size),
column_name2 data_type(size),
column_name3 data_type(size),
....
);
```

The column_name parameters specify the names of the columns of the table.

The data_type parameter specifies what type of data the column can hold (e.g. varchar, integer, decimal, date, etc.). See tables after queries examples for data types list for various DB.

The size parameter specifies the maximum length of the column of the table.

Now we want to create an empty table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City. On the W3 School website type:

```
CREATE TABLE Persons
(
PersonID int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255),
BirthDate timestamp with time zone
);
```

To rename a table on Oracle:

```
ALTER TABLE
   Persons
RENAME TO
   Employees;
```

5.4.1.1 Various SQL DB Data types

5.4.1.1.1 SQL General Data Types

Each column in a database table is required to have a name and a data type.

SQL developers have to decide what types of data will be stored inside each and every table column when creating a SQL table. The data type is a label and a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

The following table lists the general data types in SQL:

Data type	Description
CHARACTER(n)	Character string. Fixed-length n
VARCHAR(n) or CHARACTER VARYING(n)	Character string. Variable length. Maximum length n (maximum size 2000 bytes)
BINARY(n)	Binary string. Fixed-length n
BOOLEAN	Stores TRUE or FALSE values
VARBINARY(n) or BINARY VARYING(n)	Binary string. Variable length. Maximum length n
INTEGER(p)	Integer numerical (no decimal). Precision p
SMALLINT	Integer numerical (no decimal). Precision 5
INTEGER	Integer numerical (no decimal). Precision 10
BIGINT	Integer numerical (no decimal). Precision 19
DECIMAL(p,s)	Exact numerical, precision p, scale s. Example: decimal(5,2) is a number that has 3 digits before the decimal and 2 digits after the decimal
NUMERIC(p,s)	Exact numerical, precision p, scale s. (Same as DECIMAL)
FLOAT(p)	Approximate numerical, mantissa precision p. A floating number in base 10 exponential notation. The size argument for this type consists of a single number specifying the minimum precision
REAL	Approximate numerical, mantissa precision 7
FLOAT	Approximate numerical, mantissa precision 16
DOUBLE PRECISION	Approximate numerical, mantissa precision 16
DATE	Stores year, month, and day values
TIME	Stores hour, minute, and second values
TIMESTAMP	Stores year, month, day, hour, minute, and second values

INTERVAL	Composed of a number of integer fields, representing a period of time, depending on the type of interval
ARRAY	A set-length and ordered collection of elements
MULTISET	A variable-length and unordered collection of elements
XML	Stores XML data

Tableau 5 General SQL Data Types

5.4.1.1.2 Oracle 11g Data Types

String types:

Data Type	Description	Limits
char(size)	Where size is the number of characters to store. Fixed-length strings. Space padded.	Maximum size of 2000 bytes.
nchar(size)	Where size is the number of characters to store. Fixed-length NLS string Space padded.	Maximum size of 2000 bytes.
nvarchar2(size)	Where size is the number of characters to store. Variable-length NLS string.	Maximum size of 4000 bytes.
varchar2(size)	Where size is the number of characters to store. Variable-length string.	Maximum size of 4000 bytes. Maximum size of 32KB in PLSQL.
long	Variable-length strings. (backward compatible)	Maximum size of 2GB.
raw	Variable-length binary strings	Maximum size of 2000 bytes.
long raw	Variable-length binary strings. (backward compatible)	Maximum size of 2GB.

Tableau 6 Oracle 11g String Data Types

Number types:

Data Type	Description	Limits
number(p,s)	Precision can range from 1 to 38. Scale can range from -84 to 127.	Where ***p*** is the precision and ***s*** is the scale. For example, number(7,2) is a number that has 5 digits before the decimal and 2 digits after the decimal.
numeric(p,s)	Precision can range from 1 to 38.	Where ***p*** is the precision and ***s*** is the scale. For example, numeric(7,2) is a number that has 5 digits before the decimal and 2 digits after the decimal.
float		
dec(p,s)	Precision can range from 1 to 38.	Where ***p*** is the precision and ***s*** is the scale.

		For example, dec(3,1) is a number that has 2 digits before the decimal and 1 digit after the decimal.
decimal(p,s)	Precision can range from 1 to 38.	Where ***p*** is the precision and ***s*** is the scale. For example, decimal(3,1) is a number that has 2 digits before the decimal and 1 digit after the decimal.
integer		
int		
smallint		
real		
double precision		

Tableau 7 Oracle 11g Numbers Data Types

Date types:

Data Type	Description	Limits
date		A date between Jan 1, 4712 BC and Dec 31, 9999 AD.
timestamp (*fractional seconds precision*)	Includes year, month, day, hour, minute, and seconds. For example: timestamp(6)	***fractional seconds precision*** must be a number between 0 and 9. (default is 6)
timestamp (*fractional seconds precision*) with time zone	Includes year, month, day, hour, minute, and seconds; with a time zone displacement value. For example: timestamp(5) with time zone	***fractional seconds precision*** must be a number between 0 and 9. (default is 6)
timestamp (*fractional seconds precision*) with local time zone	Includes year, month, day, hour, minute, and seconds; with a time zone expressed as the session time zone. For example: timestamp(4) with local time zone	***fractional seconds precision*** must be a number between 0 and 9. (default is 6)
interval year (*year precision*) to month	Time period stored in years and months. For example: interval year(4) to month	***year precision*** is the number of digits in the year. (default is 2)
interval day (*day precision*) to second (*fractional seconds precision*)	Time period stored in days, hours, minutes, and seconds. For example: interval day(2) to second(6)	***day precision*** must be a number between 0 and 9. (default is 2) ***fractional seconds precision*** must be a number between 0 and 9. (default is 6)

Tableau 8 Oracle 11g Dates Data Types

Large objects (LOB):

Data type	Description	Storage
bfile	File locators that point to a binary file on the server file system (outside the database).	Maximum file size of 4GB.
blob	Stores unstructured binary large objects.	Store up to 4GB of binary data.
clob	Stores single-byte and multi-byte character data.	Store up to 4GB of character data.
nclob	Stores unicode data.	Store up to 4GB of character text data.

Tableau 9 Oracle 11g Large Objects Data Types

Row ID Datatypes:

Data type	Description	Storage
rowid	The format of the rowid is: BBBBBBB.RRRR.FFFFF Where BBBBBBB is the block in the database file; RRRR is the row in the block; FFFFF is the database file.	Fixed-length binary data. Every record in the database has a physical address or **rowid**.
urowid(size)		Universal rowid. Where ***size*** is optional.

Tableau 10 Oracle 11g Row ID Data Types

5.4.1.1.3 Microsoft Access Data Types

Data type	Description	Storage
Text	Use for text or combinations of text and numbers. 255 characters maximum	
Memo	Memo is used for larger amounts of text. Stores up to 65,536 characters. **Note:** You cannot sort a memo field. However, they are searchable	
Byte	Allows whole numbers from 0 to 255	1 byte
Integer	Allows whole numbers between -32,768 and 32,767	2 bytes
Long	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
Single	Single precision floating-point. Will handle most decimals	4 bytes
Double	Double precision floating-point. Will handle most decimals	8 bytes
Currency	Use for currency. Holds up to 15 digits of whole dollars, plus 4 decimal places.**Tip:** You can choose which country's currency to use	8 bytes
AutoNumber	AutoNumber fields automatically give each record its own number,	4 bytes

	usually starting at 1	
Date/Time	Use for dates and times	8 bytes
Yes/No	A logical field can be displayed as Yes/No, True/False, or On/Off. In code, use the constants True and False (equivalent to -1 and 0). **Note:** Null values are not allowed in Yes/No fields	1 bit
Ole Object	Can store pictures, audio, video, or other BLOBs (Binary Large OBjects)	up to 1GB
Hyperlink	Contain links to other files, including web pages	
Lookup Wizard	Let you type a list of options, which can then be chosen from a drop-down list	4 bytes

Tableau 11 Microsoft Access Data Types

5.4.1.1.4 MySQL Data Types

In MySQL there are three main types : text, number, and Date/Time types.

Text types:

Data type	Description
CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. **Note:** If you put a greater value than 255 it will be converted to a TEXT type
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT	Holds a string with a maximum length of 65,535 characters
BLOB	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(x,y,z,etc.)	Let you enter a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. **Note:** The values are sorted in the order you enter them. You enter the possible values in this format: ENUM('X','Y','Z')
SET	Similar to ENUM except that SET may contain up to 64 list items and can

	store more than one choice

Number types:

Data type	Description
TINYINT(size)	-128 to 127 normal. 0 to 255 UNSIGNED*. The maximum number of digits may be specified in parenthesis
SMALLINT(size)	-32768 to 32767 normal. 0 to 65535 UNSIGNED*. The maximum number of digits may be specified in parenthesis
MEDIUMINT(size)	-8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*. The maximum number of digits may be specified in parenthesis
INT(size)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis
BIGINT(size)	-9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615 UNSIGNED*. The maximum number of digits may be specified in parenthesis
FLOAT(size,d)	A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DOUBLE(size,d)	A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DECIMAL(size,d)	A DOUBLE stored as a string, allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

*The integer types have an extra option called UNSIGNED. Normally, the integer goes from an negative to positive value. Adding the UNSIGNED attribute will move that range up so it starts at zero instead of a negative number.

Date types:

Data type	Description
DATE()	A date. Format: YYYY-MM-DD **Note:** The supported range is from '1000-01-01' to '9999-12-31'
DATETIME()	*A date and time combination. Format: YYYY-MM-DD HH:MM:SS **Note:** The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP()	*A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MM:SS

	Note: The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC
TIME()	A time. Format: HH:MM:SS **Note:** The supported range is from '-838:59:59' to '838:59:59'
YEAR()	A year in two-digit or four-digit format. **Note:** Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

*Even if DATETIME and TIMESTAMP return the same format, they work very differently. In an INSERT or UPDATE query, the TIMESTAMP automatically set itself to the current date and time. TIMESTAMP also accepts various formats, like YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD, or YYMMDD.

5.4.1.1.5 SQL Server Data Types

String types:

Data type	Description	Storage
char(n)	Fixed width character string. Maximum 8,000 characters	Defined width
varchar(n)	Variable width character string. Maximum 8,000 characters	2 bytes + number of chars
varchar(max)	Variable width character string. Maximum 1,073,741,824 characters	2 bytes + number of chars
text	Variable width character string. Maximum 2GB of text data	4 bytes + number of chars
nchar	Fixed width Unicode string. Maximum 4,000 characters	Defined width x 2
nvarchar	Variable width Unicode string. Maximum 4,000 characters	
nvarchar(max)	Variable width Unicode string. Maximum 536,870,912 characters	
ntext	Variable width Unicode string. Maximum 2GB of text data	
bit	Allows 0, 1, or NULL	
binary(n)	Fixed width binary string. Maximum 8,000 bytes	
varbinary	Variable width binary string. Maximum 8,000 bytes	
varbinary(max)	Variable width binary string. Maximum 2GB	
image	Variable width binary string. Maximum 2GB	

Number types:

Data type	Description	Storage
tinyint	Allows whole numbers from 0 to 255	1 byte
smallint	Allows whole numbers between -32,768 and 32,767	2 bytes
int	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
bigint	Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	8 bytes
decimal(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
numeric(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
smallmoney	Monetary data from -214,748.3648 to 214,748.3647	4 bytes
money	Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 bytes
float(n)	Floating precision number data from -1.79E + 308 to 1.79E + 308. The n parameter indicates whether the field should hold 4 or 8 bytes. float(24) holds a 4-byte field and float(53) holds an 8-byte field. Default value of n is 53.	4 or 8 bytes
real	Floating precision number data from -3.40E + 38 to 3.40E + 38	4 bytes

Date types:

Data type	Description	Storage
datetime	From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds	8 bytes
datetime2	From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds	6-8 bytes
smalldatetime	From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute	4 bytes
date	Store a date only. From January 1, 0001 to December 31, 9999	3 bytes
time	Store a time only to an accuracy of 100 nanoseconds	3-5 bytes
datetimeoffset	The same as datetime2 with the addition of a time zone offset	8-10 bytes
timestamp	Stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real time. Each table may have only one timestamp variable	

Other data types:

Data type	Description
sql_variant	Stores up to 8,000 bytes of data of various data types, except text, ntext, and timestamp
uniqueidentifier	Stores a globally unique identifier (GUID)
xml	Stores XML formatted data. Maximum 2GB
cursor	Stores a reference to a cursor used for database operations
table	Stores a result-set for later processing

5.4.1.1.6 SQL Data Type Quick Reference

However, different databases offer different choices for the data type definition.

The following table shows some of the common names of data types between the various database platforms:

Data type	Access	SQLServer	Oracle	MySQL	PostgreSQL
boolean	Yes/No	Bit	Byte	N/A	Boolean
integer	Number (integer)	Int	Number	Int Integer	Int Integer
float	Number (single)	Float Real	Number	Float	Numeric

currency	Currency	Money	N/A	N/A	Money
string (fixed)	N/A	Char	Char	Char	Char
string (variable)	Text (<256) Memo (65k+)	Varchar	Varchar Varchar2	Varchar	Varchar
binary object	OLE Object Memo	Binary (fixed up to 8K) Varbinary (<8K) Image (<2GB)	Long Raw	Blob Text	Binary Varbinary

5.4.2 With Data Types and Constraints statements

SQL constraints are used to specify rules for the data in a table.

If there is any violation between the constraint and the data action, the action is aborted by the constraint.

Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).

SQL CREATE TABLE + CONSTRAINT Syntax:

```
CREATE TABLE table_name
(
column_name1 data_type(size) constraint_name,
column_name2 data_type(size) constraint_name,
column_name3 data_type(size) constraint_name,
....
);
```

In SQL, we have the following constraints:

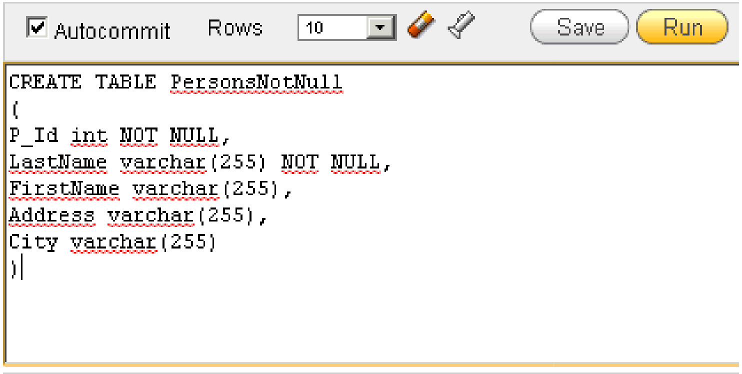
- **NOT NULL** - Indicates that a column cannot store NULL value
- **UNIQUE** - Ensures that each row for a column must have a unique value
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have an unique identity which helps to find a particular record in a table more easily and quickly
- **FOREIGN KEY** - Ensure the referential integrity of the data in one table to match values in another table
- **CHECK** - Ensures that the value in a column meets a specific condition
- **DEFAULT** - Specifies a default value when specified none for this column

The best way to study all these options is to use a real RDBMS. We will also use Oracle...!

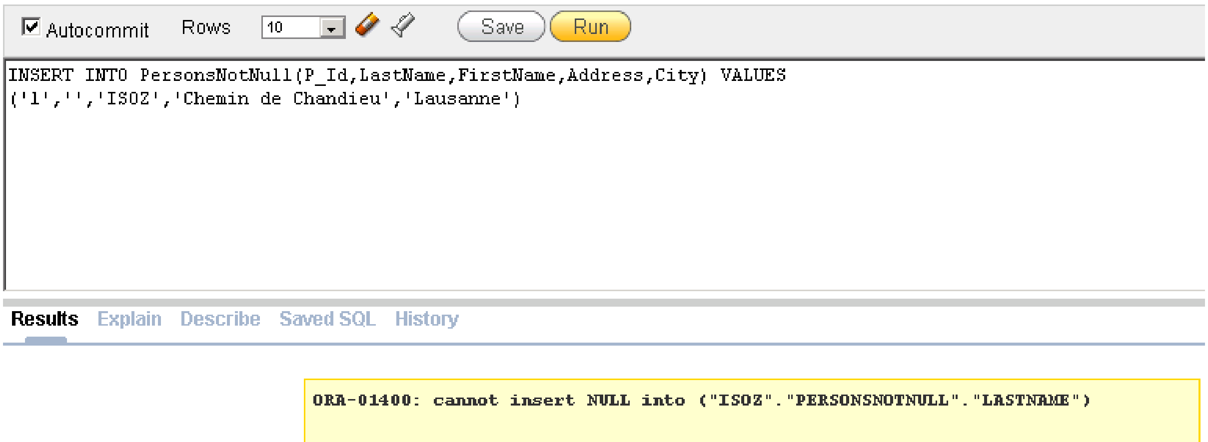
5.4.2.1 SQL NOT NULL Constraint

The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.

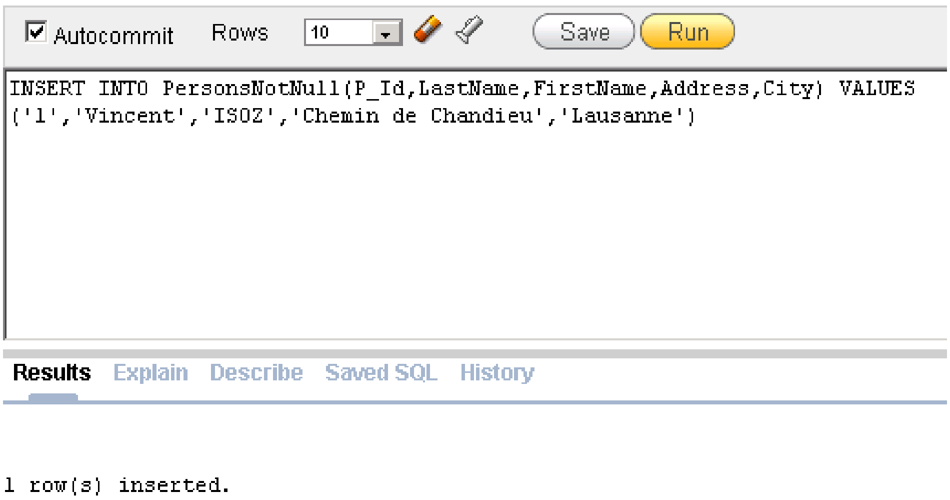
The following SQL enforces the "P_Id" column and the "LastName" column to not accept NULL values:



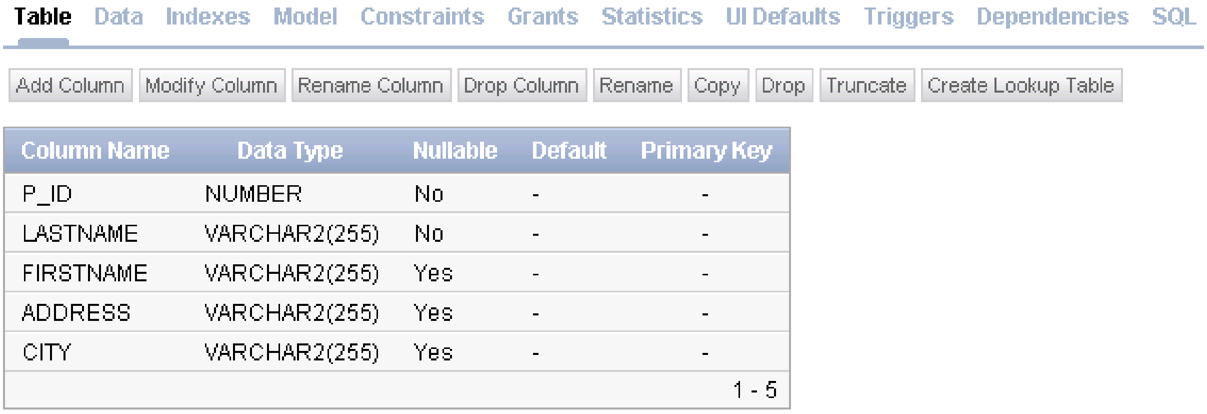
If you try then to insert a row without the LastName you will receive an error:



And if you forget nothing the operation will be successful:



as you can see in the Oracle object browser:



Column Name	Data Type	Nullable	Default	Primary Key
P_ID	NUMBER	No	-	-
LASTNAME	VARCHAR2(255)	No	-	-
FIRSTNAME	VARCHAR2(255)	Yes	-	-
ADDRESS	VARCHAR2(255)	Yes	-	-
CITY	VARCHAR2(255)	Yes	-	-
				1 - 5

5.4.2.2 SQL UNIQUE Constraint

The UNIQUE constraint uniquely identifies each record in a database table.

The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.

Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

If the creation of a UNIQUE Constraint fails this is because you already have duplicates data existing in your table in the chosen field.

5.4.2.2.1 Create a single UNIQUE constraint on table creation

The following SQL creates a UNIQUE constraint on the "P_Id" column when the "Persons" table is created:

SQL Server / Oracle / MS Access:

Autocommit Rows 10 Save Run

```
CREATE TABLE PersonsUniqueSingle
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT uP_Id UNIQUE (P_Id)
)
```

That will give:

Table Data Indexes Model **Constraints** Grants Statistics UI Defaults Triggers Dependencies SQL

Create Drop Enable Disable

Constraint	Type	Search Condition	Related Constraint	Columns	Delete Rule	Status
SYS_C007117	Check	"P_ID" IS NOT NULL	-	-	-	ENABLED
SYS_C007118	Check	"LASTNAME" IS NOT NULL	-	-	-	ENABLED
UP_ID	Unique	-	-	P_ID	-	ENABLED

MySQL:

```
CREATE TABLE PersonsUnique
(
P_Id int NOT NULL,
```

```
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
UNIQUE (P_Id)
)
```

5.4.2.2.2 Create a multiple column UNIQUE constraint on table creation

To allow naming of a UNIQUE constraint, and for defining a UNIQUE constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE PersonsUniqueMulti
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT uP_ID UNIQUE (LastName,FirstName)
)
```

5.4.2.2.3 DROP single or multiple UNIQUE constraint

To drop a single or multiple UNIQUE constraint, use the following SQL:

SQL Server / Oracle / MS Access:

```
ALTER TABLE PersonsUniqueMulti
DROP CONSTRAINT uP_ID
```

MySQL:

```
ALTER TABLE PersonsUniqueMulti
DROP INDEX uP_ID
```

5.4.2.2.4 Create a single UNIQUE constraint on an existing table

To create a UNIQUE constraint on the "P_Id" column when the table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE PersonsUniqueMulti
ADD CONSTRAINT uP_ID UNIQUE (P_Id)
```

5.4.2.2.5 Create a multiple UNIQUE constraint on an existing table

To allow naming of a UNIQUE constraint, and for defining a UNIQUE constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE PersonsUniqueMulti
ADD CONSTRAINT uP_ID UNIQUE (LastName,FirstName)
```

5.4.2.3 SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a database table.

Primary keys must contain unique values and primary key column cannot contain NULL values. Each table should also have at least one primary key.

If the creation of a PRIMARY KEY fail this is because you already have duplicates data existing in your table in the chosen field.

5.4.2.3.1 Create a single PRIMARY KEY Constraint on table creation

The following SQL creates a PRIMARY KEY on the "P_Id" column when the "Persons" table is created:

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

That will result in Oracle to:

Table Data Indexes Model **Constraints** Grants Statistics UI Defaults Triggers Dependencies SQL

Create Drop Enable Disable

Constraint	Type	Search Condition	Related Constraint	Columns	Delete Rule	Status	Last Change	Index
SYS_C007142	Check	"P_ID" IS NOT NULL	-	-	-	ENABLED	09/16/2013 09:07:55 PM	-
SYS_C007143	Check	"LASTNAME" IS NOT NULL	-	-	-	ENABLED	09/16/2013 09:07:55 PM	-
SYS_C007144	Primary	-	-	P_ID	-	ENABLED	09/16/2013 09:07:55 PM	SYS_C007144

1 - 3

as you can see this result in an horrible Index Name. The better is then to use:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT pkPerson PRIMARY KEY (P_Id)
)
```

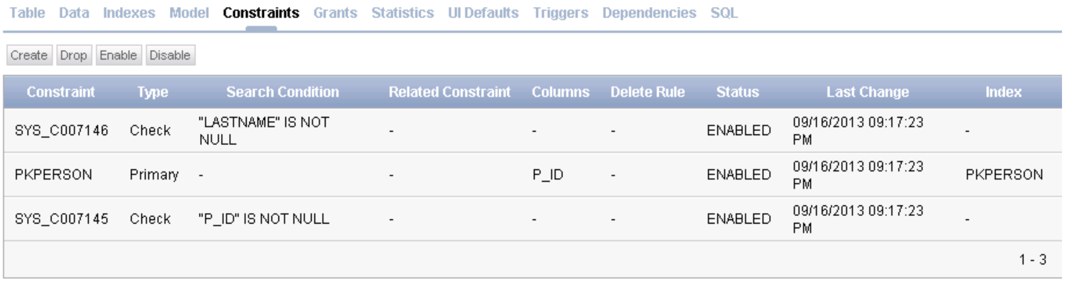


Table Data Indexes Model **Constraints** Grants Statistics UI Defaults Triggers Dependencies SQL

Create Drop Enable Disable

Constraint	Type	Search Condition	Related Constraint	Columns	Delete Rule	Status	Last Change	Index
SYS_C007146	Check	"LASTNAME" IS NOT NULL	-	-	-	ENABLED	09/16/2013 09:17:23 PM	-
PKPERSON	Primary	-	-	P_ID	-	ENABLED	09/16/2013 09:17:23 PM	PKPERSON
SYS_C007145	Check	"P_ID" IS NOT NULL	-	-	-	ENABLED	09/16/2013 09:17:23 PM	-

1 - 3

MySQL:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
PRIMARY KEY (P_Id)
)
```

5.4.2.3.2 Create a multiple PRIMARY KEY Constraint on table creation

The following SQL creates a PRIMARY KEY on the "LastName" and "FirstName" columns when the "Persons" table is created:

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
CONSTRAINT pkPerson PRIMARY KEY (LastName,FirstName)
)
```

That will result in Oracle to:

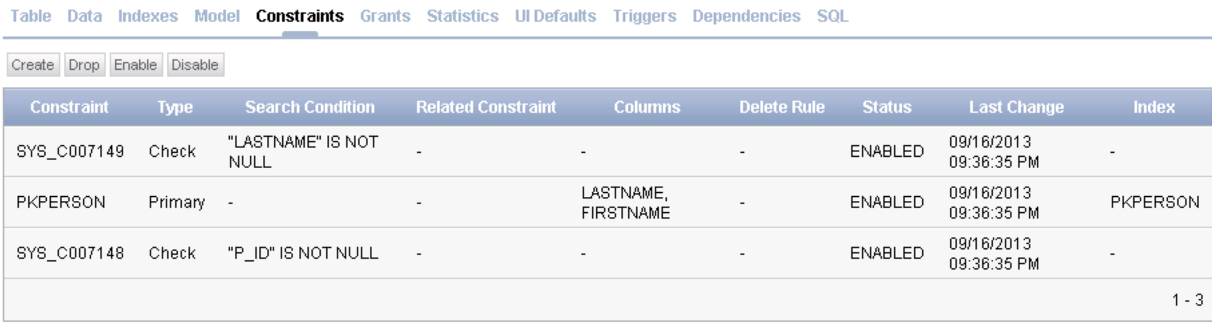


Table Data Indexes Model **Constraints** Grants Statistics UI Defaults Triggers Dependencies SQL

Create Drop Enable Disable

Constraint	Type	Search Condition	Related Constraint	Columns	Delete Rule	Status	Last Change	Index
SYS_C007149	Check	"LASTNAME" IS NOT NULL	-	-	-	ENABLED	09/16/2013 09:36:35 PM	-
PKPERSON	Primary	-	-	LASTNAME, FIRSTNAME	-	ENABLED	09/16/2013 09:36:35 PM	PKPERSON
SYS_C007148	Check	"P_ID" IS NOT NULL	-	-	-	ENABLED	09/16/2013 09:36:35 PM	-

1 - 3

MySQL:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
```

```
FirstName varchar(255),
Address varchar(255),
City varchar(255),
PRIMARY KEY (LastName,FirstName)
)
```

5.4.2.3.3 DROP single or multiple PRIMARY KEY Constraint

To drop a PRIMARY KEY constraint, use the following SQL:

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT pkPerson
```

MySQL:

```
ALTER TABLE Persons
DROP PRIMARY KEY
```

5.4.2.3.4 Create a single PRIMARY KEY constraint on an existing table

To create a PRIMARY KEY constraint on the "P_Id" column when the table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CONSTRAINT pkPerson PRIMARY KEY (P_Id)
```

5.4.2.3.5 Create a multiple PRIMARY KEY constraint on an existing table

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CONSTRAINT pkPerson PRIMARY KEY (LastName,FirstName)
```

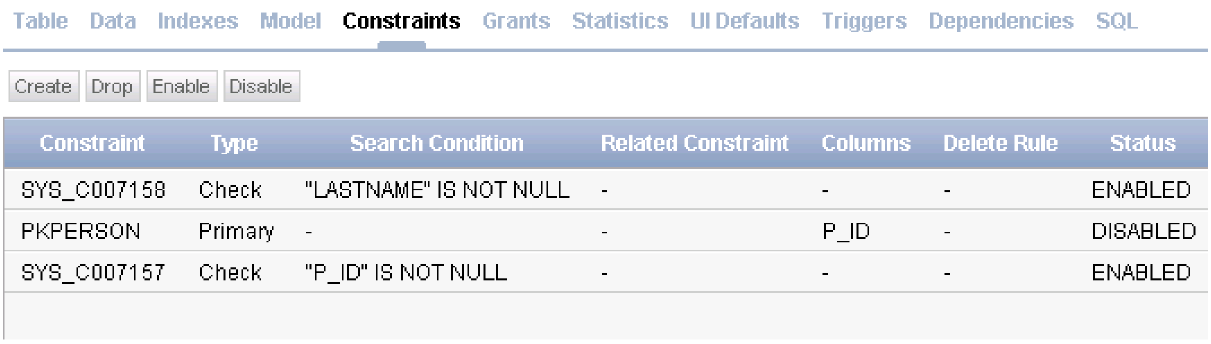
5.4.2.3.6 DISABLE/ENABLE single or multiple PRIMARY KEY Constraint

To disable a PRIMARY KEY constraint (**or any other constraint**) to speed up deletion or addition of a huge amount of data, use the following SQL:

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DISABLE CONSTRAINT pkPerson
```

This will give:



Constraint	Type	Search Condition	Related Constraint	Columns	Delete Rule	Status
SYS_C007158	Check	"LASTNAME" IS NOT NULL	-	-	-	ENABLED
PKPERSON	Primary	-	-	P_ID	-	DISABLED
SYS_C007157	Check	"P_ID" IS NOT NULL	-	-	-	ENABLED

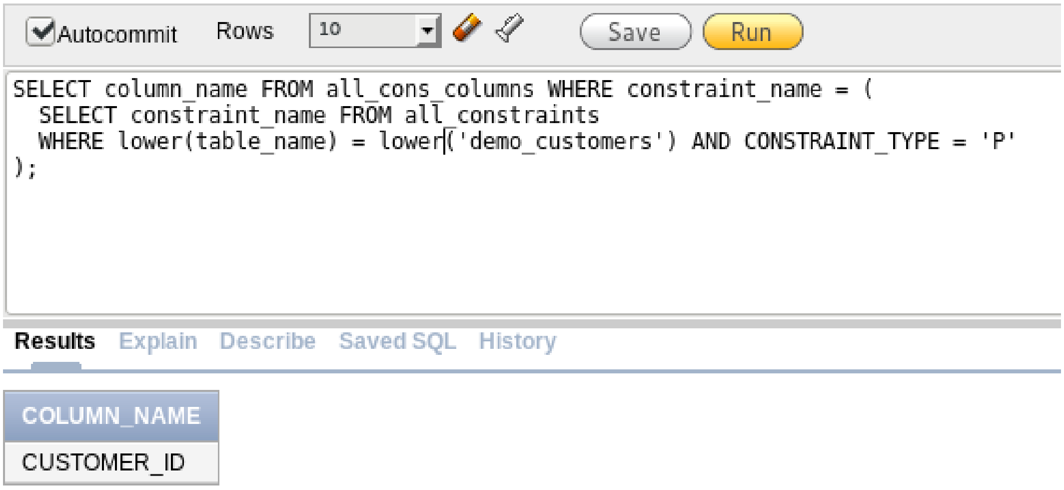
you can't with Oracle without PL/SQL disable multiple constraints. With SQL Server there is a nice query to disable all at once (see on Google).

To reactivate a constraint, we will use:

```
ALTER TABLE Persons
ENABLE CONSTRAINT pkPerson
```

5.4.2.3.7 List all primary keys from a table

It may happen that sometimes you want to get the list of all indexes of a table. To do this use the Oracle all_constraints reserved word of Oracle:




```
SELECT column_name FROM all_cons_columns WHERE constraint_name = (
  SELECT constraint_name FROM all_constraints
  WHERE lower(table_name) = lower('demo_customers') AND CONSTRAINT_TYPE = 'P'
);
```

COLUMN_NAME
CUSTOMER_ID

5.4.2.4 SQL FOREIGN KEY Constraint

A FOREIGN KEY in one table points to a PRIMARY KEY in another table as seen in the database theory training.

Let's illustrate the foreign key with an example for Oracle.

5.4.2.4.1 Create a single FOREIGN KEY Constraint on table creation

We want a table to know what is the fidelity card number of a given customer and which employee (saler) sold the card.

To do this we will run the follwing SQL in Oracle (this code must also work for mySQL, Access and others...):

Autocommit Rows 10 Save Run

```
CREATE TABLE Demo_FidelityCard
(
FidelityCard_id int NOT NULL,
CardNumber int NOT NULL,
fkCustomer_Id int NOT NULL,
fkSaler_Id int NOT NULL,
CONSTRAINT fkCustomer FOREIGN KEY (fkCustomer_Id) REFERENCES Demo_Customers(Customer_Id),
CONSTRAINT fkSaler FOREIGN KEY (fkSaler_Id) REFERENCES Demo_Users(User_Id)
)
```

This will give:

Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependencies SQL

Add Column Modify Column Rename Column Drop Column Rename Copy Drop Truncate Create Lookup Table

Column Name	Data Type	Nullable	Default	Primary Key
FIDELITYCARD_ID	NUMBER	No	-	1
CARDNUMBER	NUMBER	No	-	-
FKCUSTOMER_ID	NUMBER	No	-	-
FKSALER_ID	NUMBER	No	-	-
				1 - 4

with:

Table Data Indexes Model **Constraints** Grants Statistics UI Defaults Triggers Dependencies SQL

Create Drop Enable Disable

Constraint	Type	Search Condition	Related Constraint	Columns	Delete Rule	Status
SYS_C007188	Check	"CARDNUMBER" IS NOT NULL	-	-	-	ENABLED
SYS_C007189	Check	"FKCUSTOMER_ID" IS NOT NULL	-	-	-	ENABLED
SYS_C007190	Check	"FKSALER_ID" IS NOT NULL	-	-	-	ENABLED
PKFIDELITYCARD	Primary	-	-	FIDELITYCARD_ID	-	ENABLED
FKCUSTOMER	Foreign	-	ISOZ.DEMO_CUSTOMERS.DEMO_CUSTOMERS_PK	FKCUSTOMER_ID	NO ACTION	ENABLED
FKSALER	Foreign	-	ISOZ.DEMO_USERS.DEMO_USERS_PK	FKSALER_ID	NO ACTION	ENABLED
SYS_C007187	Check	"FIDELITYCARD_ID" IS NOT NULL	-	-	-	ENABLED

and:

Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers **Dependencies** SQL

References

Owner	Table Name
ISOZ	DEMO_USERS
ISOZ	DEMO_CUSTOMERS

Referenced by

This table is not referenced by other objects.

Synonyms

No synonyms found.

Now if you try to inset the following:

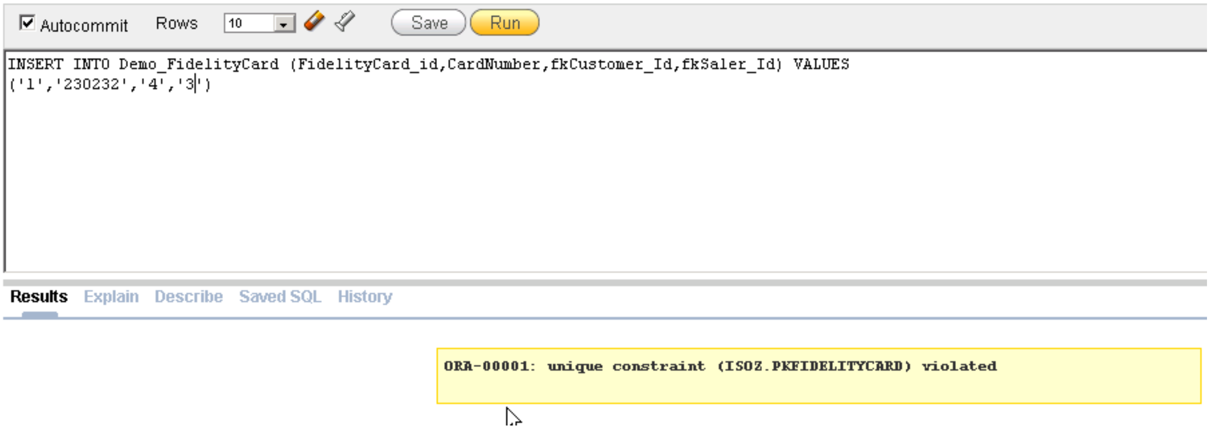
Autocommit Rows 10 Save Run

```
INSERT INTO Demo_FidelityCard (FidelityCard_id,CardNumber,fkCustomer_Id,fkSaler_Id) VALUES
('1','230232','4','1')
```

Results Explain Describe Saved SQL History

```
1 row(s) inserted.
```

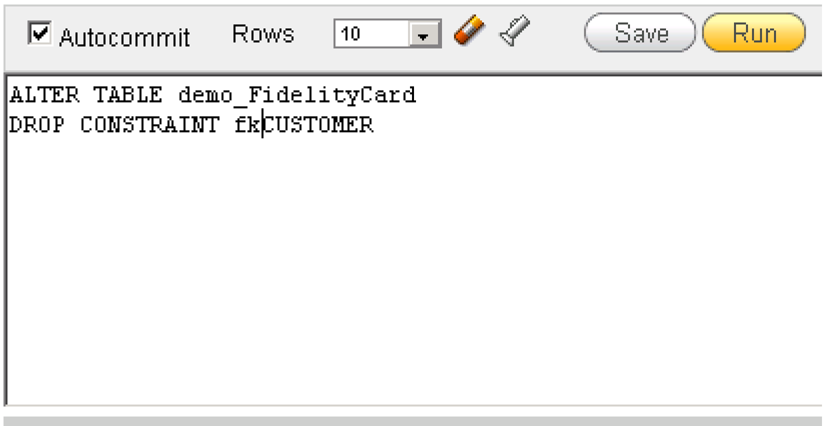
It will succeed because Customer ID4 and Saler ID 1 exists but:



will fail because Saler ID 3 does not exist!

5.4.2.4.2 DROP FOREIGN KEY Constraint

To drop a foreign key on Oracle (SQL Server/Access) you use:

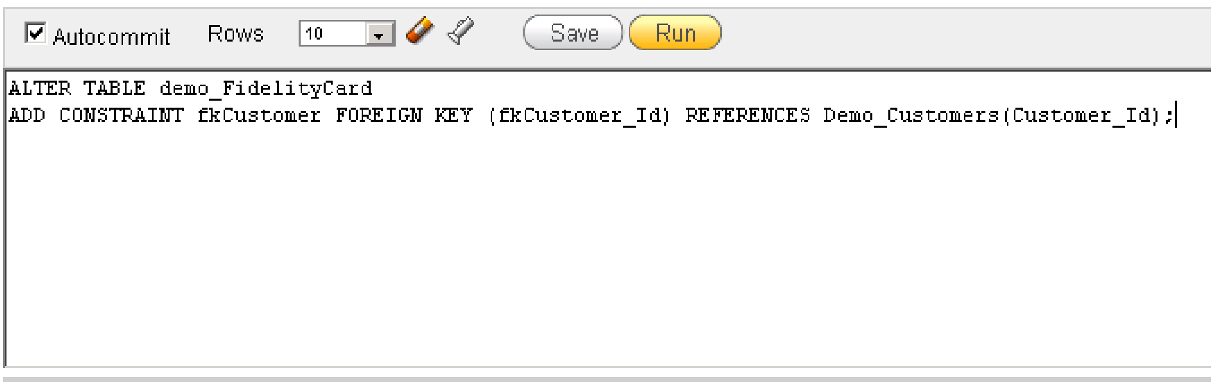


on MySQL:

```
ALTER TABLE demo_FidelityCard
DROP CONSTRAINT KEY fkCustomer
```

5.4.2.4.3 Create a FOREIGN KEY constraint on an existing table

For Oracle (also SQL Server, MySQL, Access and others):

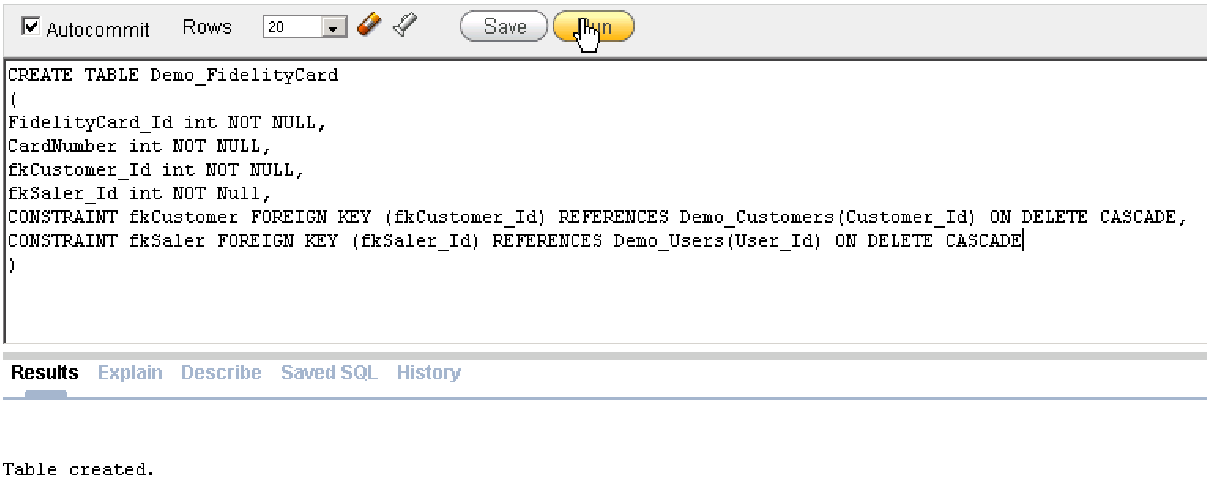


5.4.2.4.4 Foreign Key with ON DELETE CASCADE

A foreign key with a cascade delete means that if a record in the parent table is deleted, then the corresponding records in the child table with automatically be deleted. This is called a cascade delete.

A foreign key with a cascade deletion can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

Here is an example. First, we create our table:



Then you can try... If you delete a customer, the related FidelityCard will be removed. Same thing if you remove only the sale!

5.4.2.5 SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

5.4.2.5.1 Create a single or multiple CHECK Constraint on table creation

We want to create a card fidelity table where Fidelity Card Number must all be greather than 1'000'000 and at the same time accept only Sales who's ID is greather than 1 (the same syntax should also work on MySQL, SQL Server, MS Access...):

Autocommit Rows 10 Save Run

```
CREATE TABLE Demo_FidelityCard
(
FidelityCard_Id int NOT NULL,
CardNumber int NOT NULL,
fkCustomer_Id int NOT NULL,
fkSaler_Id int NOT NULL,
CONSTRAINT pkFidelityCard PRIMARY KEY (FidelityCard_Id),
CONSTRAINT fkCustomer FOREIGN KEY (fkCustomer_id) REFERENCES Demo_Customers(Customer_Id),
CONSTRAINT fkSaler FOREIGN KEY (fkSaler_Id) REFERENCES Demo_Users(User_Id),
CONSTRAINT chk_CardNumberAndSales CHECK (fkSaler_Id>1 AND CardNumber>1000000)
)
```

This will give:

Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependencies SQL

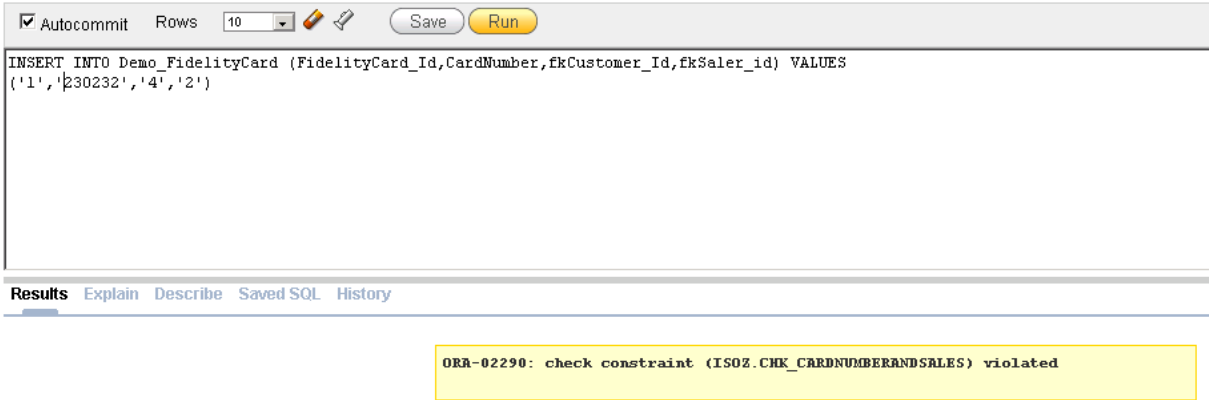
Add Column Modify Column Rename Column Drop Column Rename Copy Drop Truncate Create Lookup Table

Column Name	Data Type	Nullable	Default	Primary Key
FIDELITYCARD_ID	NUMBER	No	-	1
CARDNUMBER	NUMBER	No	-	-
FKCUSTOMER_ID	NUMBER	No	-	-
FKSALER_ID	NUMBER	No	-	-
				1 - 4

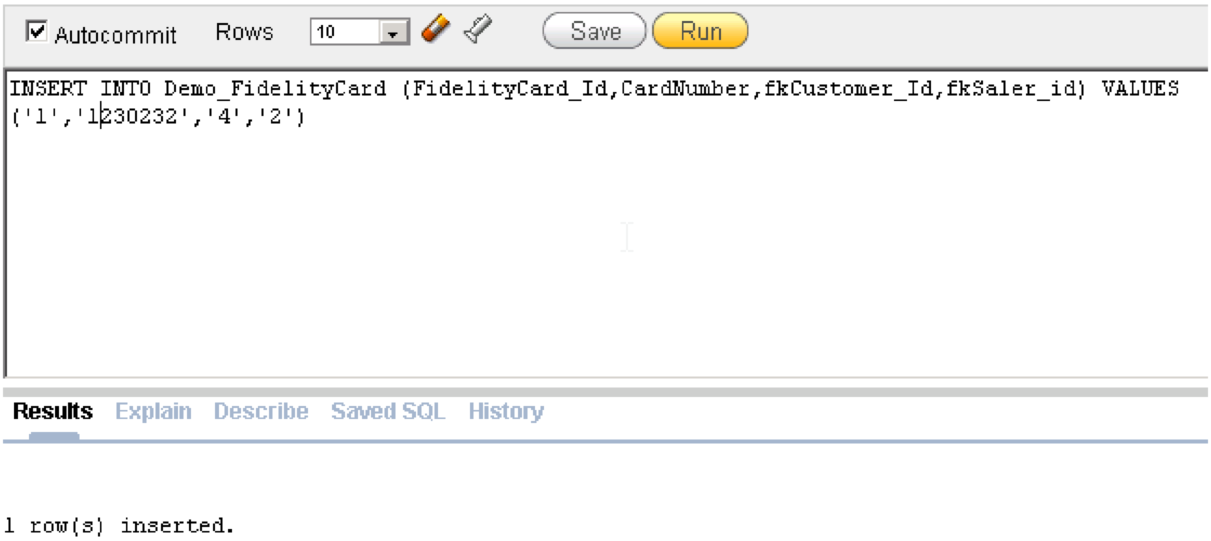
And:

Constraint	Type	Search Condition	Related Constraint	Columns	Delete Rule	Status
SYS_C007209	Check	"CARDNUMBER" IS NOT NULL	-	-	-	ENABLED
SYS_C007210	Check	"FKCUSTOMER_ID" IS NOT NULL	-	-	-	ENABLED
SYS_C007211	Check	"FKSALER_ID" IS NOT NULL	-	-	-	ENABLED
CHK_CARDNUMBERANDSALES	Check	fkSaler_Id>1 AND CardNumber>1000000	-	-	-	ENABLED
PKFIDELITYCARD	Primary	-	-	FIDELITYCARD_ID	-	ENABLED
FKCUSTOMER	Foreign	-	ISOZ.DEMO_CUSTOMERS.DEMO_CUSTOMERS_PK	FKCUSTOMER_ID	NO ACTION	ENABLED
FKSALER	Foreign	-	ISOZ.DEMO_USERS.DEMO_USERS_PK	FKSALER_ID	NO ACTION	ENABLED
SYS_C007208	Check	"FIDELITYCARD_ID" IS NOT NULL	-	-	-	ENABLED

And if you try to insert the following:



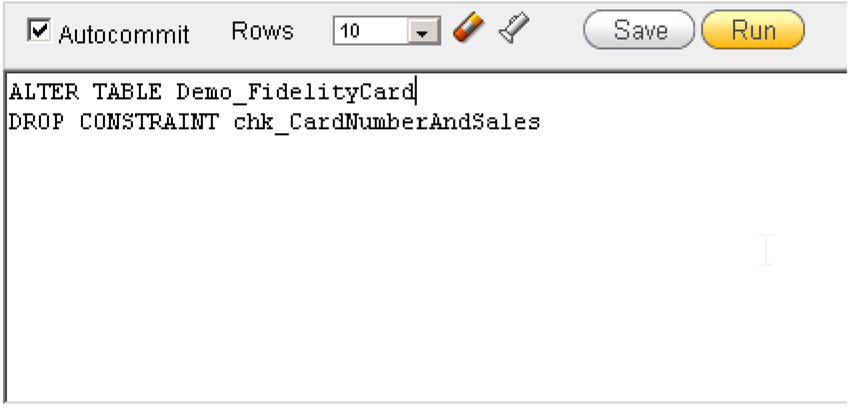
And if you respect all constraints you will get:



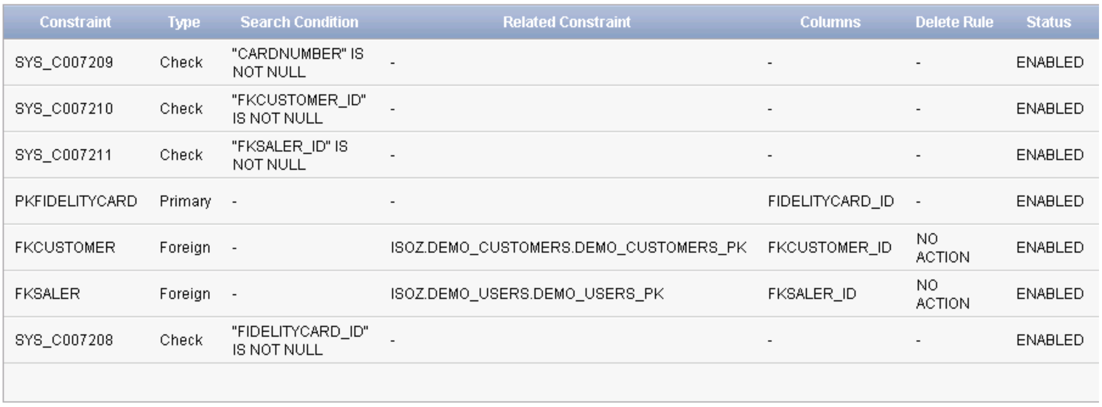
with success!

5.4.2.5.2 DROP CHECK Constraint

To drop a check just write:



and when you run the SQL code you will see the Check removed:



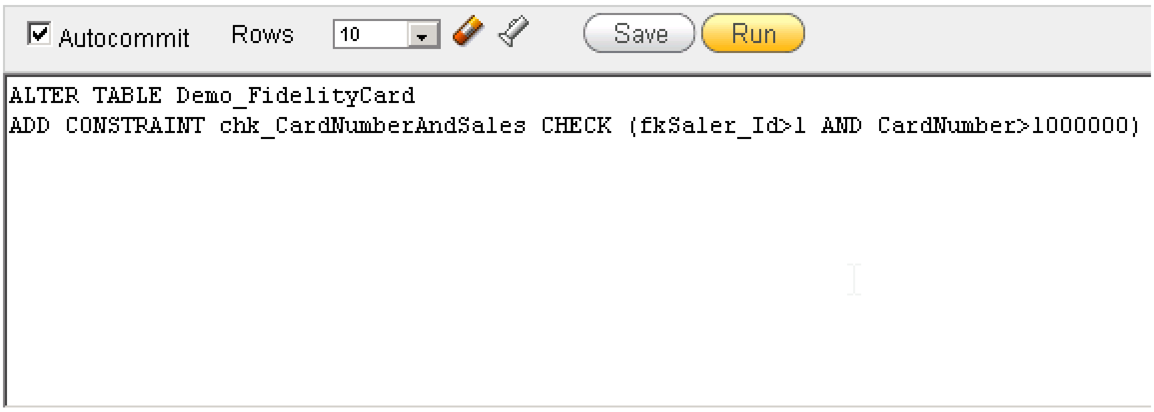
Constraint	Type	Search Condition	Related Constraint	Columns	Delete Rule	Status
SYS_C007209	Check	"CARDNUMBER" IS NOT NULL	-	-	-	ENABLED
SYS_C007210	Check	"FKCUSTOMER_ID" IS NOT NULL	-	-	-	ENABLED
SYS_C007211	Check	"FKSALER_ID" IS NOT NULL	-	-	-	ENABLED
PKFIDELITYCARD	Primary	-	-	FIDELITYCARD_ID	-	ENABLED
FKCUSTOMER	Foreign	-	ISOZ.DEMO_CUSTOMERS.DEMO_CUSTOMERS_PK	FKCUSTOMER_ID	NO ACTION	ENABLED
FKSALER	Foreign	-	ISOZ.DEMO_USERS.DEMO_USERS_PK	FKSALER_ID	NO ACTION	ENABLED
SYS_C007208	Check	"FIDELITYCARD_ID" IS NOT NULL	-	-	-	ENABLED

on MySQL the syntax is:

```
ALTER TABLE Demo_FidelityCard
DROP CHECK chk_CardNumberAndSales
```

5.4.2.5.3 Create CHECK constraint on an existing table

To add a CHECK constraint on most RDBMS the syntax is:



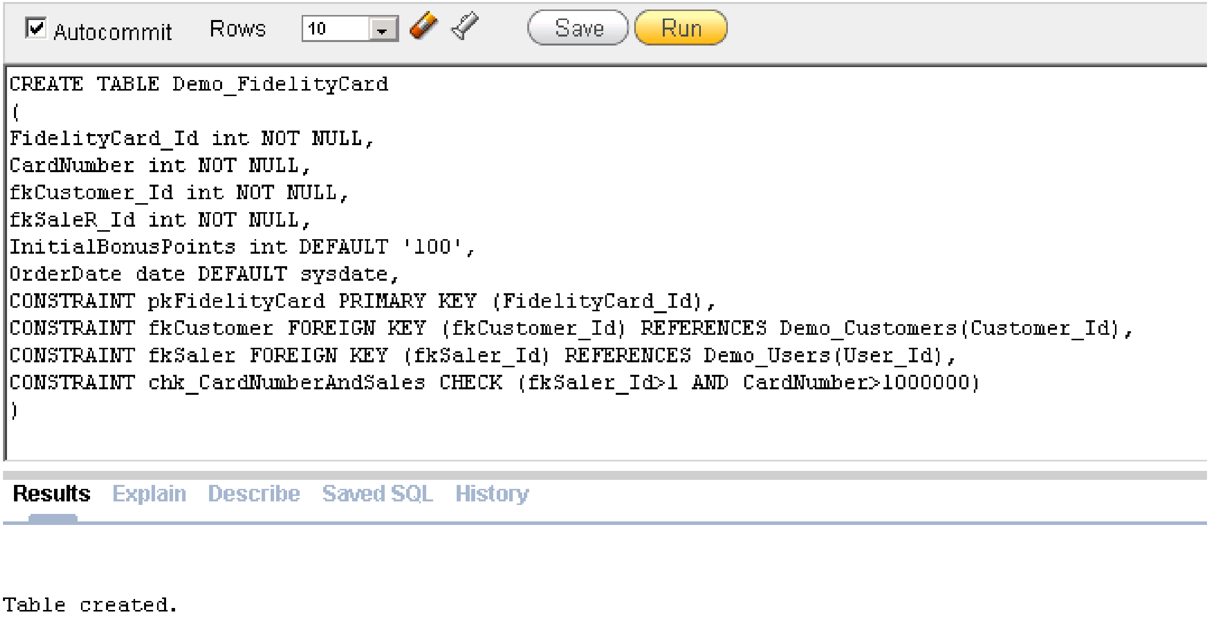
5.4.2.6 SQL DEFAULT Value

The DEFAULT constraint is used to insert a default value into a column.

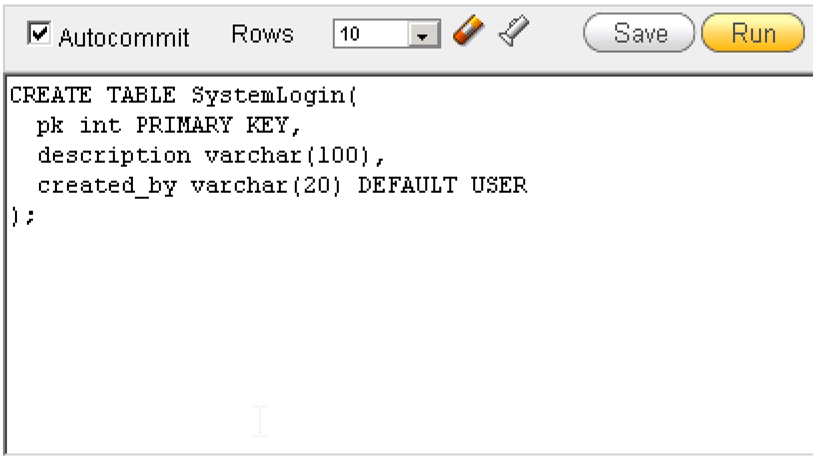
The default value will be added to all new records, if no other value is specified.

5.4.2.6.1 Create a Default Value on table creation

Once again, we will play with Oracle:

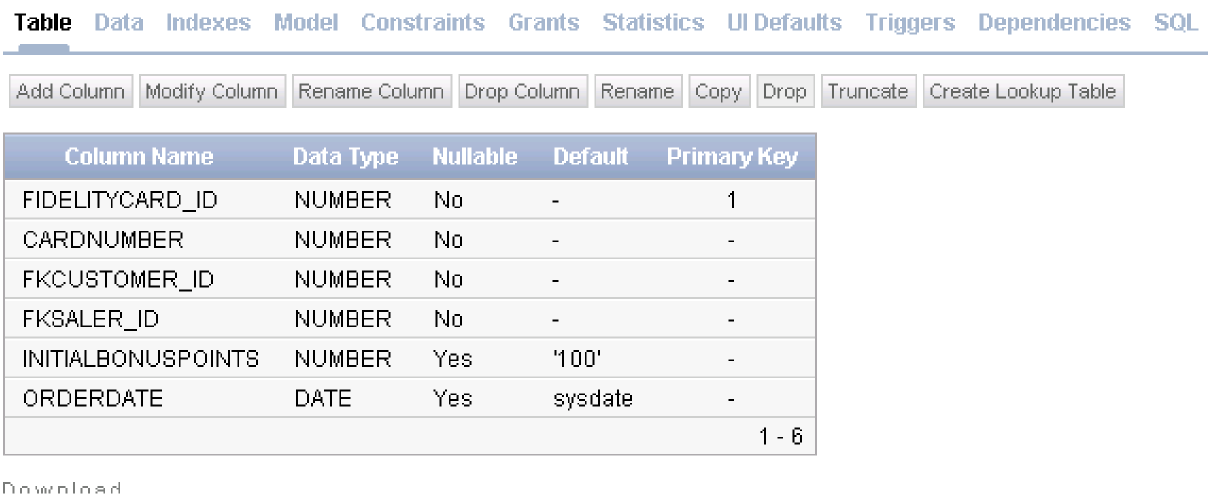


where you can see the important SYSDATE statement used a lot also sometimes with the USER statement!



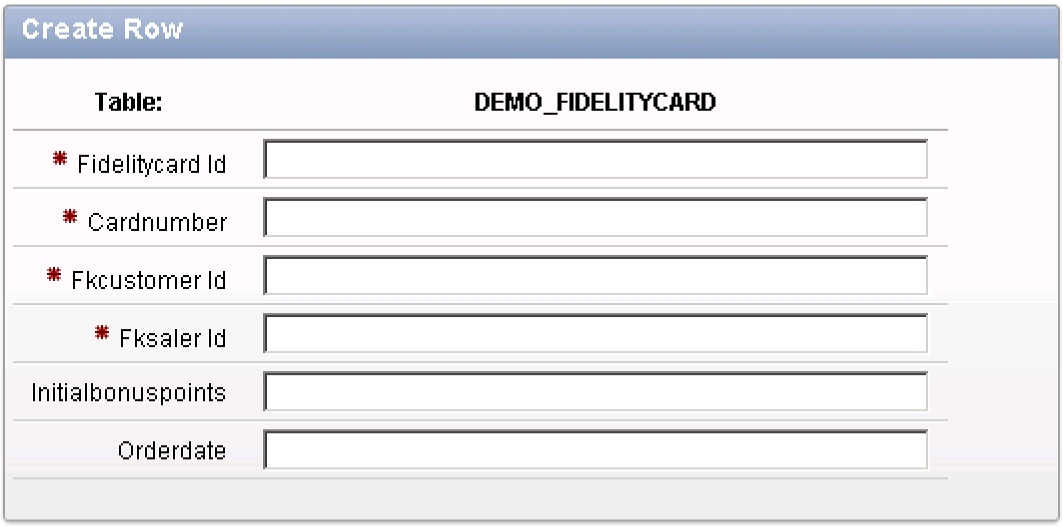
Note: On mySQL, Access, SQL Server you have to replace the **sysdate** with **getdate**().

This first query (the query that interest us) gives:

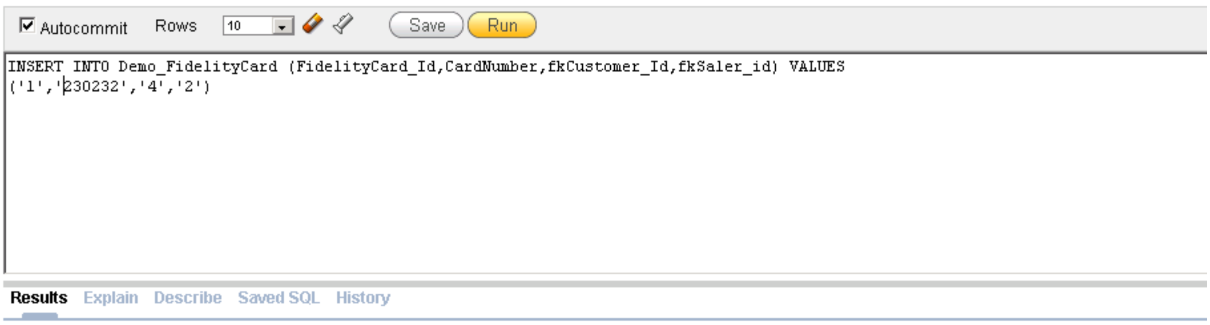


Column Name	Data Type	Nullable	Default	Primary Key
FIDELITYCARD_ID	NUMBER	No	-	1
CARDNUMBER	NUMBER	No	-	-
FKCUSTOMER_ID	NUMBER	No	-	-
FKSALER_ID	NUMBER	No	-	-
INITIALBONUSPOINTS	NUMBER	Yes	'100'	-
ORDERDATE	DATE	Yes	sysdate	-
				1 - 6

But if we use the GUI to insert rows, the standard values do not appear:

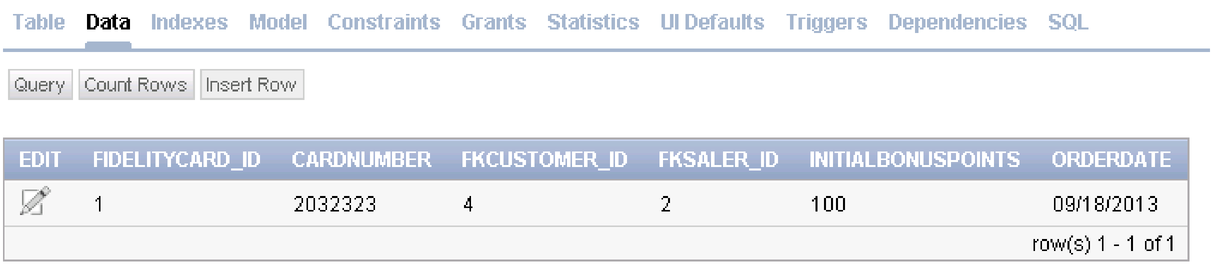


But if we insert using SQL:



INSERT INTO Demo_FidelityCard (FidelityCard_Id,CardNumber,fkCustomer_Id,fkSaler_Id) VALUE ('1','2334323','4','2')

we get:

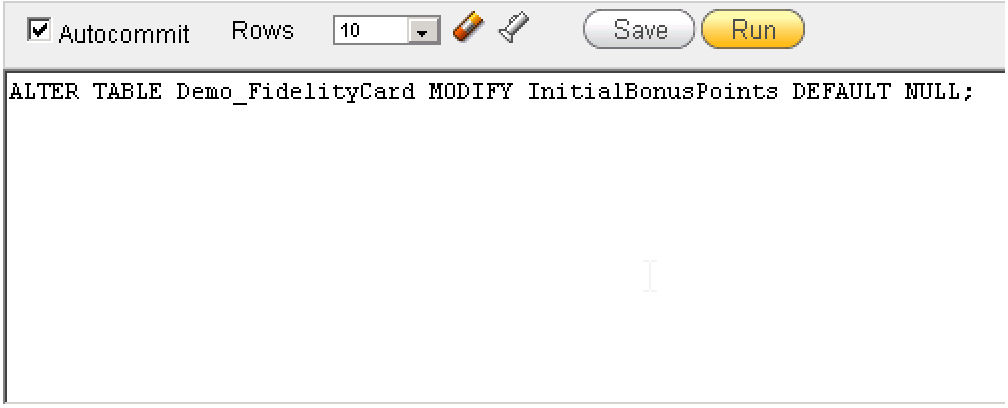


EDIT	FIDELITYCARD_ID	CARDNUMBER	FKCUSTOMER_ID	FKSALER_ID	INITIALBONUSPOINTS	ORDERDATE
	1	2032323	4	2	100	09/18/2013
						row(s) 1 - 1 of 1

... it works!

5.4.2.6.2 DROP Default Value Constraint

To drop a default value on Oracle:



To drop a DEFAULT constraint, use the following SQL on other RDBMS:

MySQL:

```
ALTER TABLE Demo_FidelityCard
ALTER InitialBonusPoints DROP DEFAULT
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Demo_FidelityCard
ALTER COLUMN InitialBonusPoints DROP DEFAULT
```

5.4.2.6.3 Create a Default Value on an existing table

To create a DEFAULT constraint on the "InitialBonusPoints" column when the table is already created, use the following SQL:

MySQL:

```
ALTER TABLE Demo_FidelityCard
ALTER InitialBonusPoints SET DEFAULT '100'
```

SQL Server / MS Access:

```
ALTER TABLE Demo_FidelityCard
ALTER InitialBonusPoints City SET DEFAULT '100'
```

Oracle:

```
ALTER TABLE Demo_FidelityCard
MODIFY InitialBonusPoints DEFAULT '100'
```

5.4.2.7 SQL CREATE INDEX statement Value

Much more about indexes with the Enterprise version of Oracle:

http://docs.oracle.com/cd/B19306_01/server.102/b14231/indexes.htm

The CREATE INDEX statement is used to create indexes in tables.

Indexes allow the database application to find data fast; without reading the whole table.

The users cannot see the indexes, they are mainly just used to speed up searches/queries.

Indexes are normally created only and only if the users say that the database begins to retrieve information too slowly. Create them only after table creation and on users requests otherwise you use disk space for nothing!

If the creation of a UNIQUE INDEX fails this is because you already have duplicates data existing in your table in the chosen field.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So you should only create indexes on columns (and tables) that will be frequently searched against.

Creates an index on a table. Duplicate values are allowed:

```
CREATE INDEX index_name
ON table_name (column_name)
```

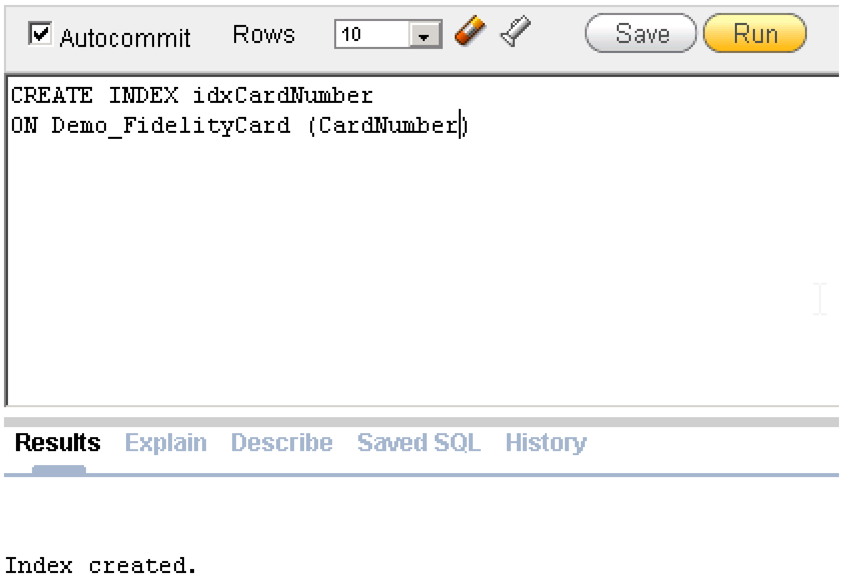
Creates a unique index on a table. Duplicate values are not allowed:

```
CREATE UNIQUE INDEX index_name
ON table_name (column_name)
```

Note: The syntax for creating indexes varies amongst different databases. Therefore: Check the syntax for creating indexes in your database.

5.4.2.7.1 Create a Single (aka non-clustered) Nonunique Index on an existing table

Once again, we will play with Oracle:

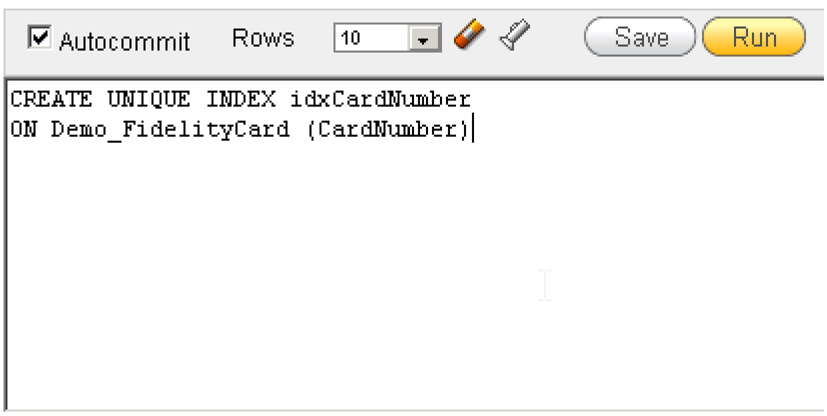


This will give:

Index Name	Uniqueness	Columns	Status	Index Type	Temporary	Partitioned	Function Status	Join Index
IDXCARDNUMBER	NONUNIQUE	CARDNUMBER	VALID	NORMAL	N	NO	-	NO
							1 - 1	

5.4.2.7.2 Create a Single (aka non-clustered) Unique Index on an existing table

Duplicate values will not be allowed. It's like creating a Nonunique Index and after putting and UNIQUE constraint on it:



This will give:

Index Name	Uniqueness	Columns	Status	Index Type	Temporary	Partitioned	Function Status	Join Index
IDXCARDNUMBER	UNIQUE	CARDNUMBER	VALID	NORMAL	N	NO	-	NO

1 - 1

It is easier to manage than creating and Nonunique INDEX with after a UNIQUE CONSTRAINT.

Note: On MS Access, when you create a Primary Key, on unique Index is automatically created on the primary key column.

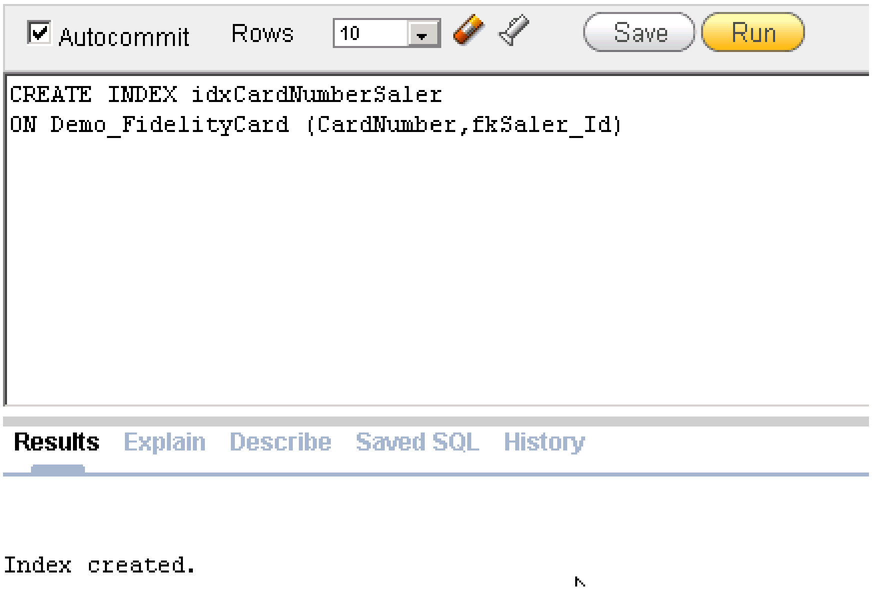
5.4.2.7.3 Create a Multiple (aka clustered) Nonunique Index on an existing table

If an employee uses a lot of queries using only 'CardNumber' field creating a non-clustered index is for sure the efficient answer. But if you have another employee using a lot of time queries using 'CardNumber' and 'fkSaler' then it will be interesting to create a clustered Index.

Depending on the scenario and storage availability and also update frequency of the table you can have cluster index on the 2-uplet ('CardNumber','fkSaler') + two index on respectively the same fields.

The best solution is not always easy. The best thing is to study usage statistics and compare results using statistical tools (student T-test typically).

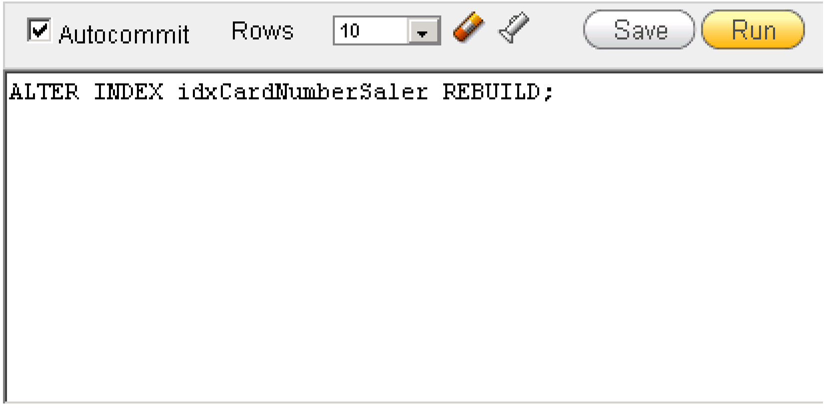
To create a multiple (clustered) non-unique Index on Oracle on an existing table use the following:



and for sure you can also create a multiple (clustered) UNIQUE INDEX.

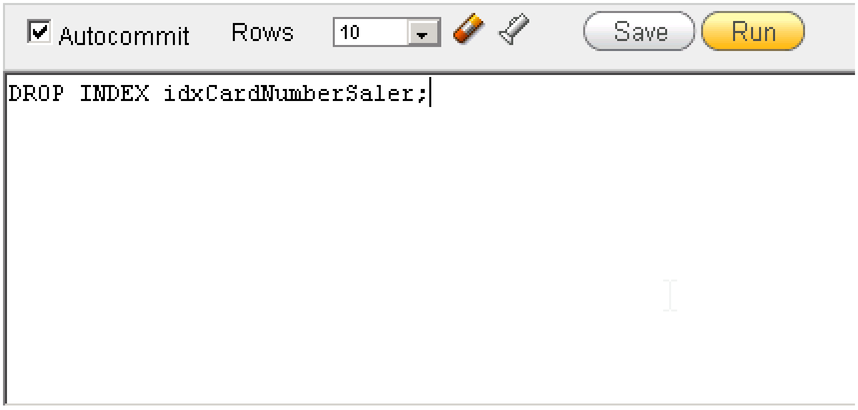
5.4.2.7.4 Rebuild an Index

An index can be corrupted on the tree needs to be optimized again. To rebuild and Index, run the following on Oracle:



5.4.2.7.5 DROP Multiple/Single Unique/Nonunique Index

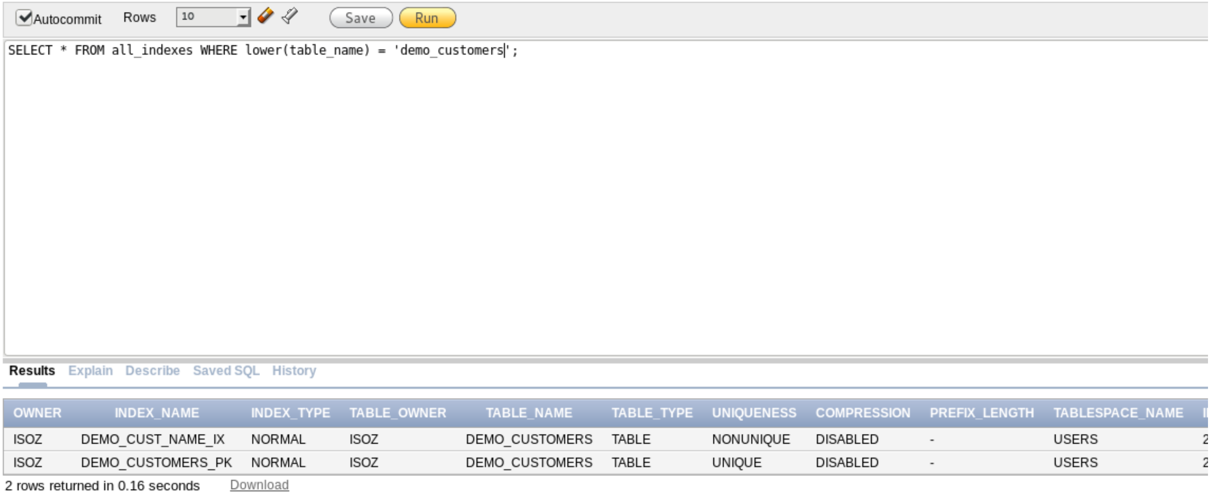
To drop an INDEX on Oracle you just write:



You don't need to specify the table because index names are unique across the whole server.

5.4.2.7.6 List all indexes from a table

It may happen that sometimes you want to get the list of all indexes of a table. To do this use the Oracle all_indexes reserved word:



5.5 SQL ALTER TABLE Statement

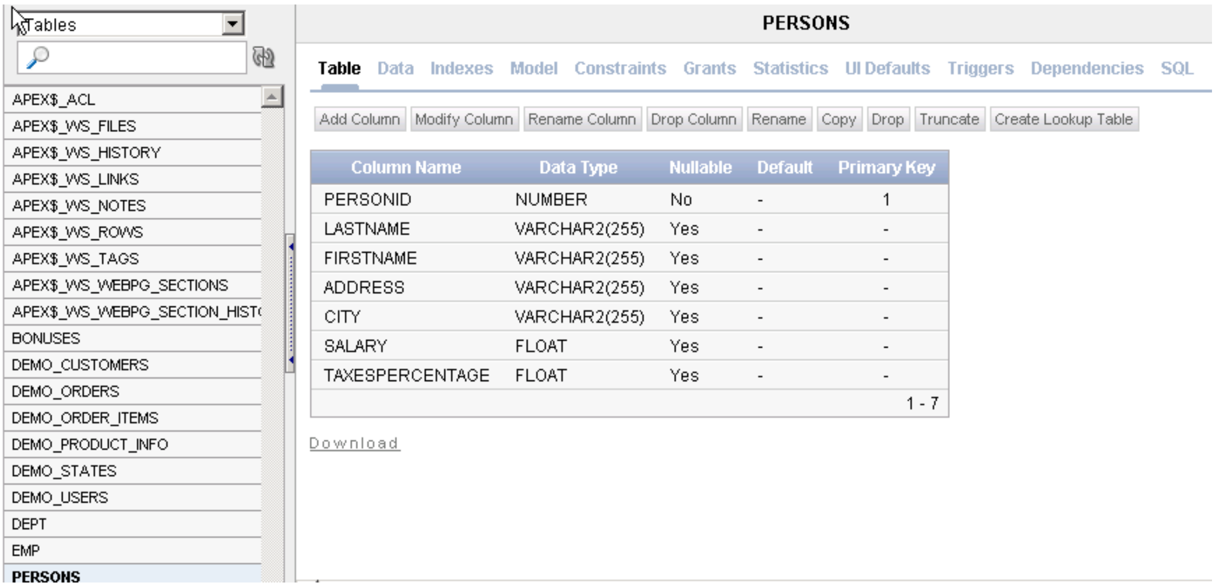
Here is a resume of some new ALTER TABLE statements and some other we already know (all examples are given only for Oracle):

5.5.1 ALTER TABLE to change table name

First, we create in Oracle the table:

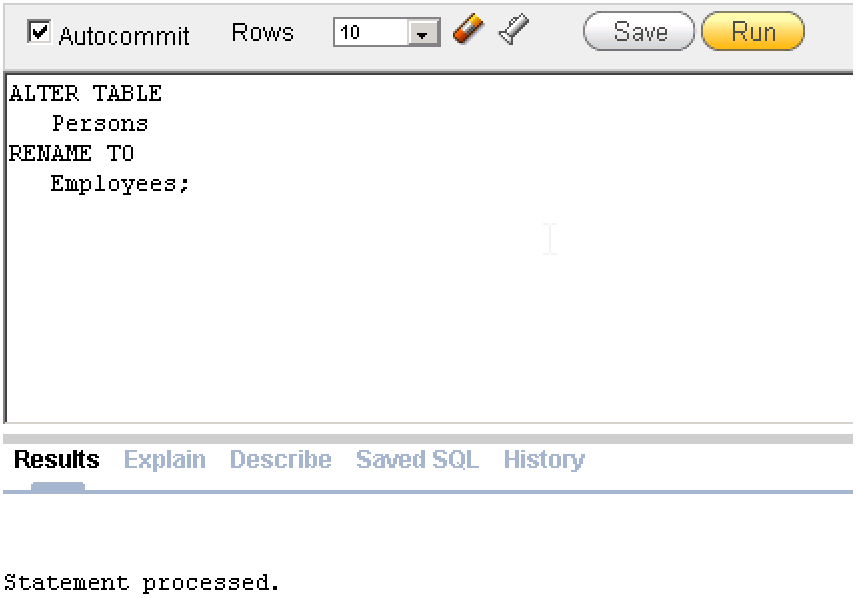
```
CREATE TABLE Persons
(
PersonID int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255),
Salary float,
TaxesPercentage float,
CONSTRAINT pkPerson PRIMARY KEY (PersonID)
);
```

We get:



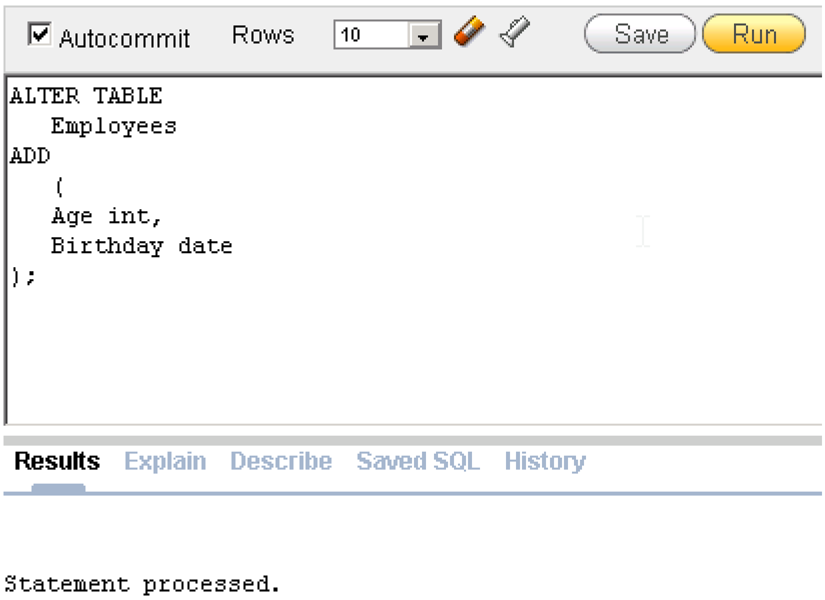
Column Name	Data Type	Nullable	Default	Primary Key
PERSONID	NUMBER	No	-	1
LASTNAME	VARCHAR2(255)	Yes	-	-
FIRSTNAME	VARCHAR2(255)	Yes	-	-
ADDRESS	VARCHAR2(255)	Yes	-	-
CITY	VARCHAR2(255)	Yes	-	-
SALARY	FLOAT	Yes	-	-
TAXESPERCENTAGE	FLOAT	Yes	-	-

and after we rename it :



5.5.2 ALTER TABLE to add (static) new column

To alter a table to add columns:

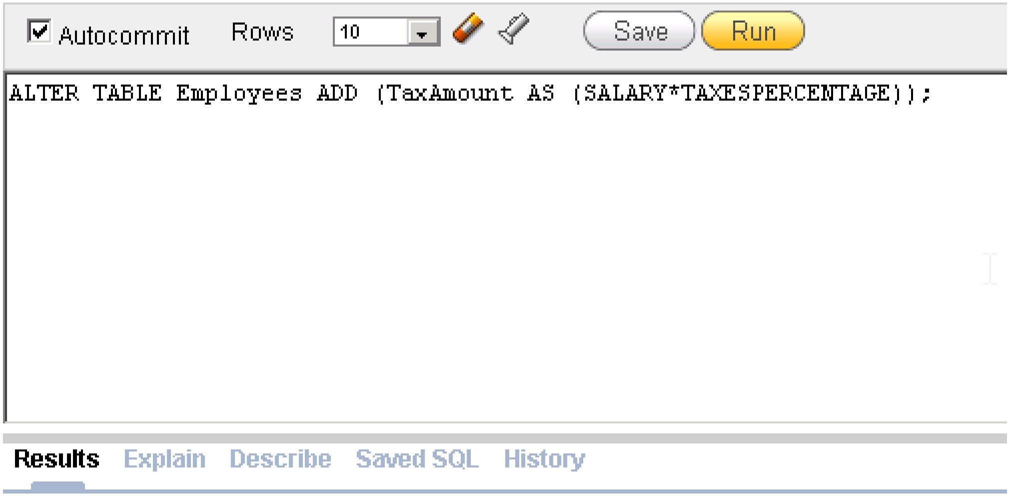


5.5.3 ALTER TABLE to add virtual (dynamic) new column

Computed columns are nothing new to Oracle and have been available since its first release in 1984. A special type of column - known as a computed by column - defines a calculation instead of a data type. This special column takes no space within the table but allows the programmer to fetch the value at run-time using the select statement, or via a cursor.

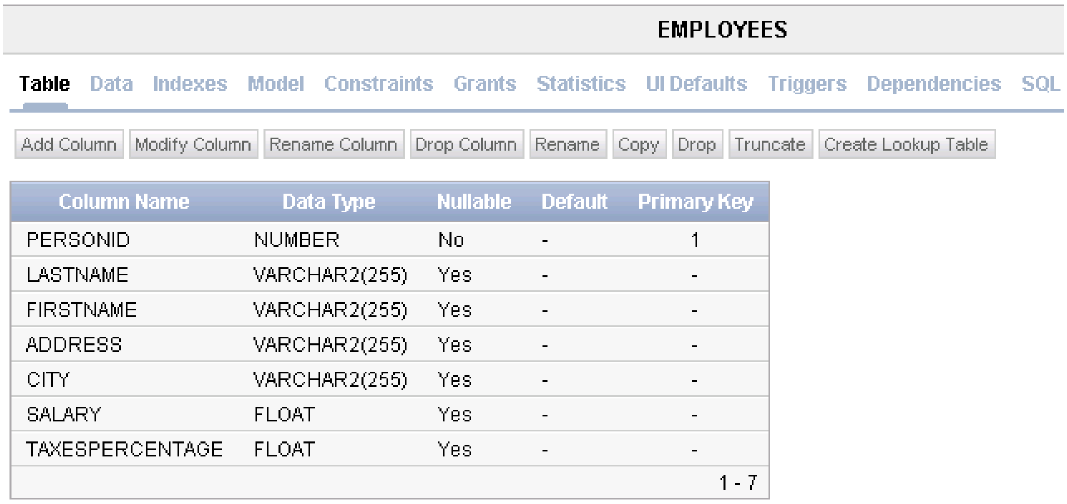
The computed by expression can be based only on pure functions!!

Use the ALTER TABLE statement to add AUTOMATIC new column.



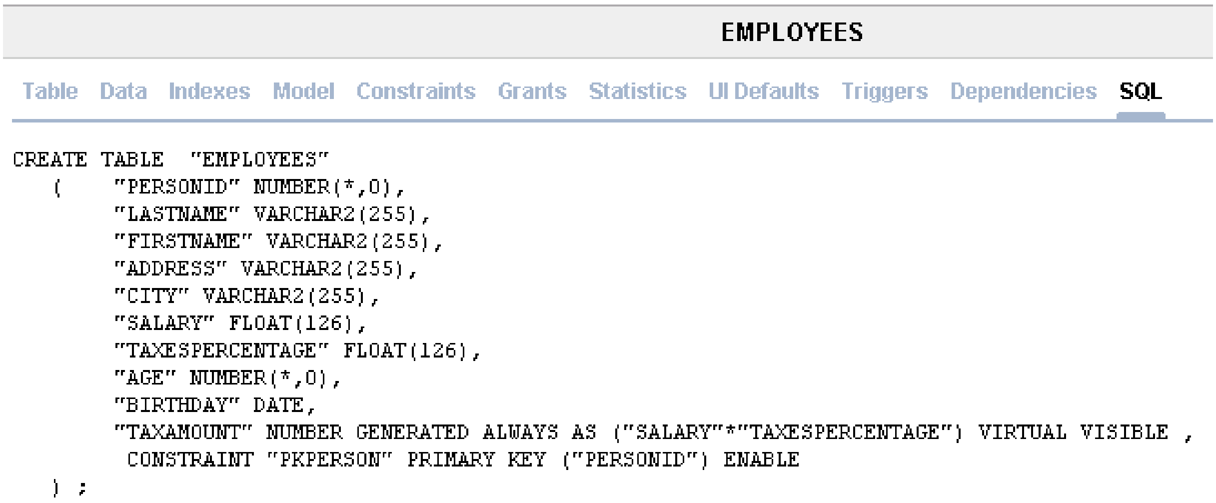

```
Table altered.
```

If we look at the table structure we get:

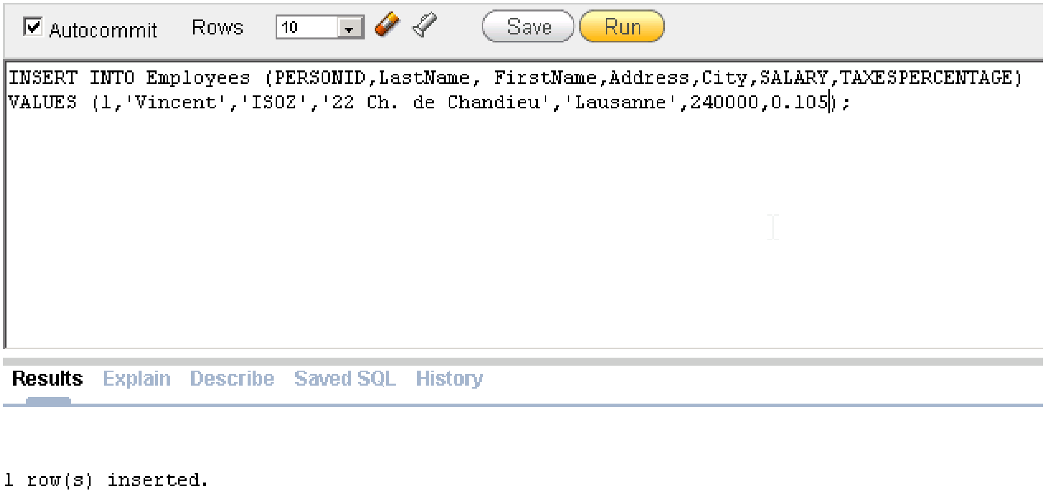


Column Name	Data Type	Nullable	Default	Primary Key
PERSONID	NUMBER	No	-	1
LASTNAME	VARCHAR2(255)	Yes	-	-
FIRSTNAME	VARCHAR2(255)	Yes	-	-
ADDRESS	VARCHAR2(255)	Yes	-	-
CITY	VARCHAR2(255)	Yes	-	-
SALARY	FLOAT	Yes	-	-
TAXESPERCENTAGE	FLOAT	Yes	-	-
				1 - 7

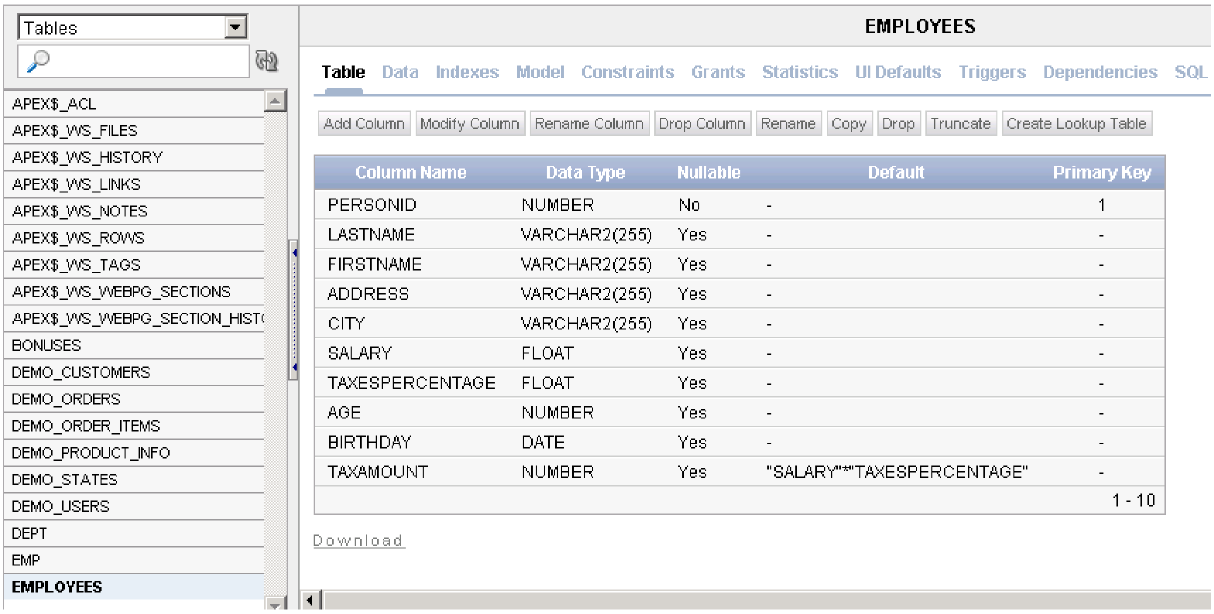
as you can see the virtual column is not visible in the table structure but if we look in the SQL structure, we can see TaxAmount:



Now if we add a new row:

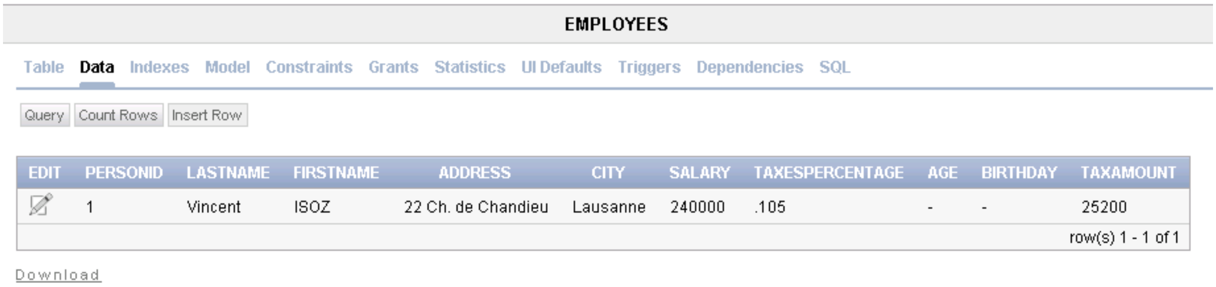


Now that a least one row exists, we have:



Column Name	Data Type	Nullable	Default	Primary Key
PERSONID	NUMBER	No	-	1
LASTNAME	VARCHAR2(255)	Yes	-	-
FIRSTNAME	VARCHAR2(255)	Yes	-	-
ADDRESS	VARCHAR2(255)	Yes	-	-
CITY	VARCHAR2(255)	Yes	-	-
SALARY	FLOAT	Yes	-	-
TAXESPERCENTAGE	FLOAT	Yes	-	-
AGE	NUMBER	Yes	-	-
BIRTHDAY	DATE	Yes	-	-
TAXAMOUNT	NUMBER	Yes	"SALARY"*"TAXESPERCENTAGE"	-

and we can look at the content:



EDIT	PERSONID	LASTNAME	FIRSTNAME	ADDRESS	CITY	SALARY	TAXESPERCENTAGE	AGE	BIRTHDAY	TAXAMOUNT
	1	Vincent	ISOZ	22 Ch. de Chandieu	Lausanne	240000	.105	-	-	25200

it works!

5.5.4 ALTER TABLE to change column name

To change a column name just use the following syntax:

```
ALTER TABLE Employees RENAME COLUMN Birthday to BirthDate;
```

5.5.5 ALTER TABLE to change column type

To change a column type just use the following syntax:

```
ALTER TABLE
   Employees
MODIFY
(
   FirstName varchar(30),
   LastName varchar(30)
);
```

5.5.6 ALTER TABLE to change Constraints name

The following SQL code can be used to change the name of a **Primary Key, a Foreign Key, an Index or a Unique constraint**:

```
ALTER TABLE
   Employees
RENAME CONSTRAINT
(
   pkPerson TO pkPersonId
);
```

5.5.7 ALTER TABLE to change Index name

First create an index on our table:

```
CREATE INDEX idxFirstName ON Employees (FirstName);
```

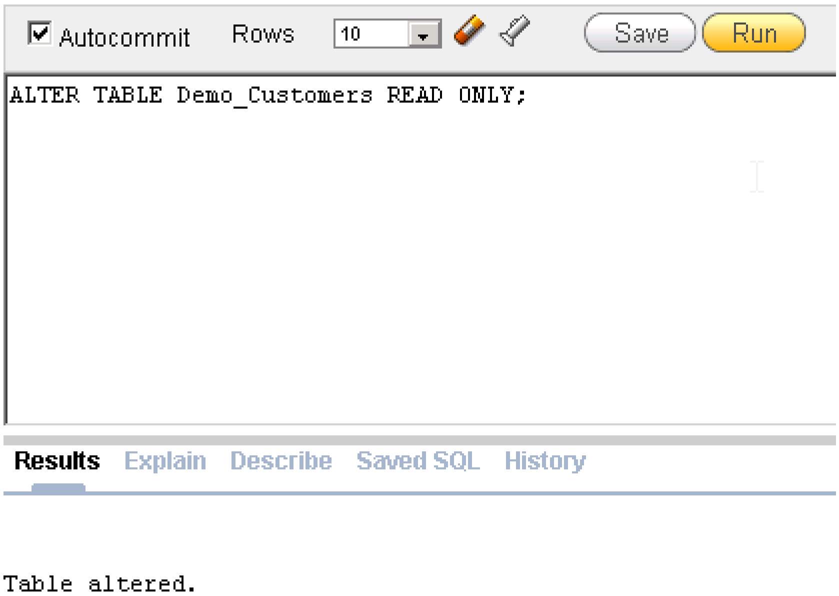
And to change the name of the index:

```
ALTER INDEX idxFirstName RENAME TO idxFName;
```

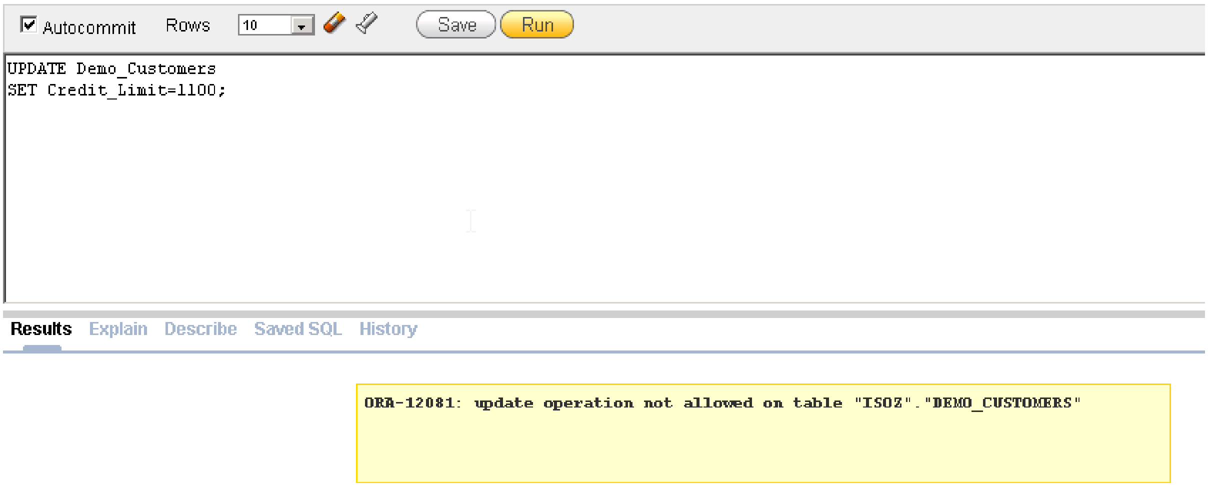
5.5.8 ALTER TABLE to change table in Read Only

Sometimes you will need to protect tables against DML from end-users. Then the best solution could be to protect de table in read only to avoid any data modification.

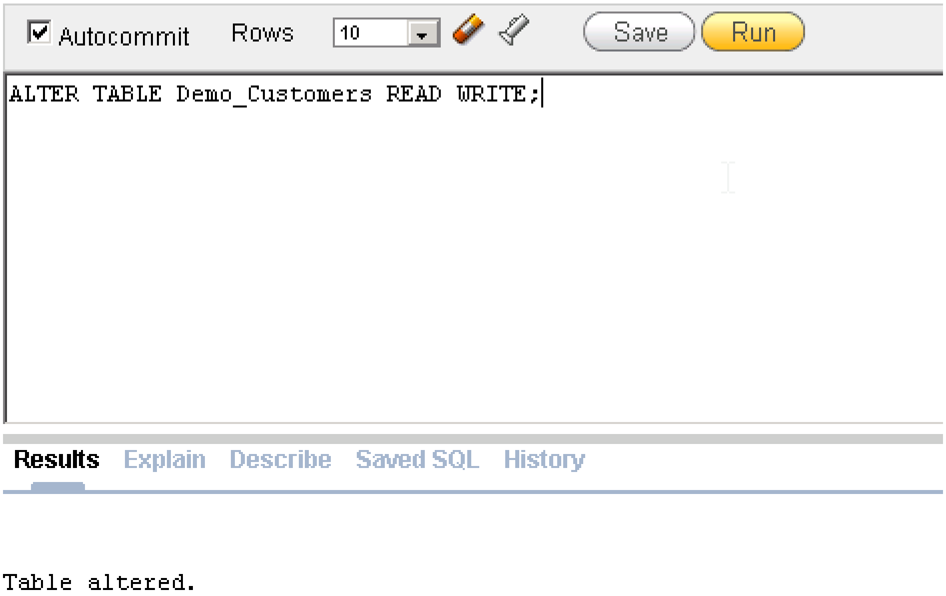
To do this run the following code in Oracle:



And now if you try to run and DML query you will get an error:



and if you change it again in READ/WRITE you will be able to run the DML:



5.6 SQL DROP Statement

Indexes, tables, columns and databases can easily be deleted/removed with the DROP:

5.6.1 Drop a database

To drop a database we won't make a practical because this can be done with a simple right clic on Access, SQL Server and can't be done with Oracle and on the free version of MySQL this statement is blocked.

Then to remove a database, when you have the rights and the possibility, the syntax is simply:

```
DROP DATABASE database_name
```

5.6.2 Drop a table

The DROP TABLE statement is used to delete a table.

```
DROP TABLE table_name;
```

5.6.3 Drop column(s)

To drop a column, you have to alter the table:

```
ALTER TABLE
   table_name
DROP
   (col_name1, col_name2);
```

5.6.3.1 UNUSED column(s)

If you are concerned about the length of time it could take to drop column data from all of the rows in a large table, you can use the ALTER TABLE...SET UNUSED statement. This statement marks one or more columns as unused, but does not actually remove the target column data or restore the disk space occupied by these columns. However, a column that is marked as unused is not displayed in queries or data dictionary views, and its name is removed so that a new column can reuse that name. All constraints, indexes, and statistics defined on the column are also removed.

```
ALTER TABLE
   table_name
SET UNUSED
   (col_name1, col_name2);
```

You can later remove columns that are marked as unused by issuing an ALTER TABLE...DROP UNUSED COLUMNS statement. Unused columns are also removed from the target table whenever an explicit drop of any particular column or columns of the table is issued.

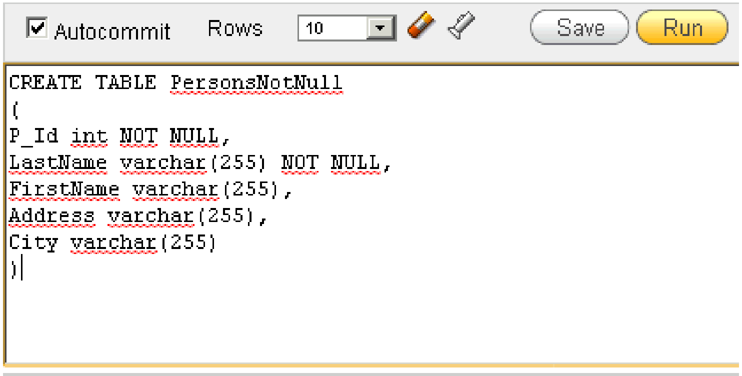
```
ALTER TABLE
   table_name
DROP UNUSED
   (col_name1, col_name2);
```

It is no longer possible to retrieve marked columns when clearing a table to make them operational again. Only the DROP UNUSED COLUMNS directive is allowed to handle such columns. It destroys all the columns of a table that are marked at erasure.

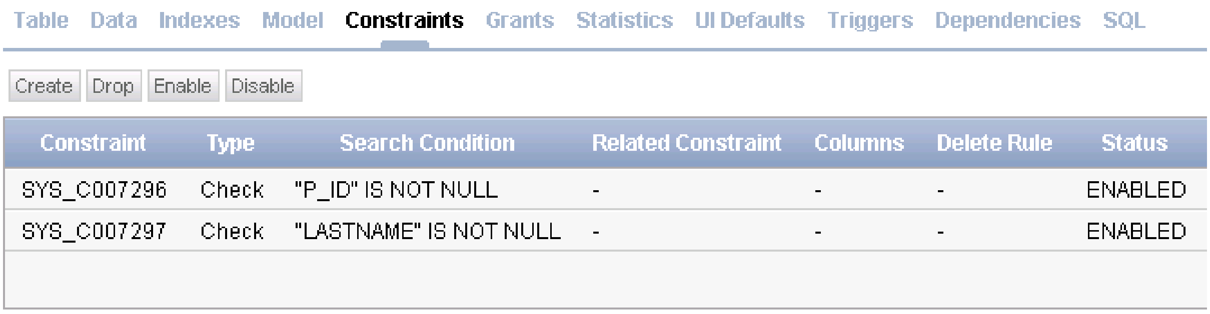
5.6.4 Drop constraints

We will focus here only on Oracle SQL and with a NOT NULL constraint example (the idea is the same for **Primary Key, a Foreign Key, an Index or a Unique constraint**).

To see this with a NOT NULL we create first a table:

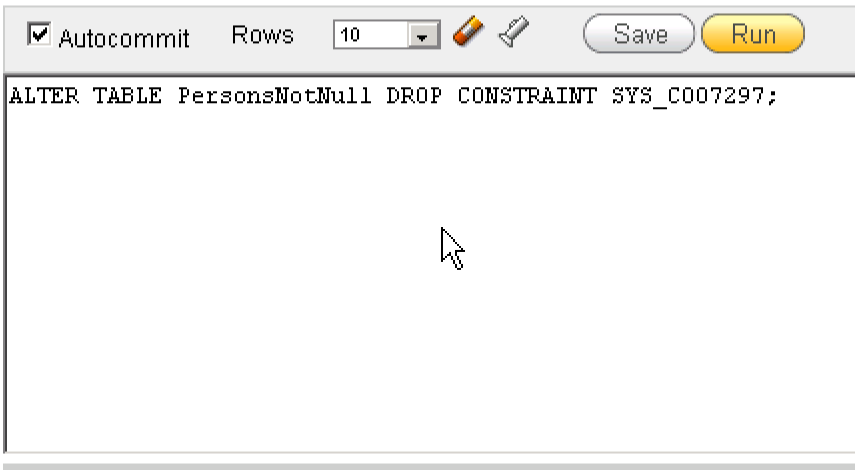


Then you will see that NOT NULL is only a constraint:



Constraint	Type	Search Condition	Related Constraint	Columns	Delete Rule	Status
SYS_C007296	Check	"P_ID" IS NOT NULL	-	-	-	ENABLED
SYS_C007297	Check	"LASTNAME" IS NOT NULL	-	-	-	ENABLED

The if you know how to remove a constraint you know how to remove and NOT NULL. For this you just type:



5.6.5 Drop index

The DROP INDEX statement is used to delete an index in a table.

DROP INDEX Syntax for MS Access:

```
DROP INDEX index_name ON table_name
```

DROP INDEX Syntax for MS SQL Server:

```
DROP INDEX table_name.index_name
```

DROP INDEX Syntax for DB2/Oracle (you do not need to specify table name because index name are unique across the whole server):

```
DROP INDEX index_name
```

DROP INDEX Syntax for MySQL:

```
ALTER TABLE table_name DROP INDEX index_name
```

5.6.6 Drop the content of a table

What if we only want to delete the data inside the table, and not the table itself?

Then, use the TRUNCATE TABLE statement:

```
TRUNCATE TABLE table_name
```

5.7 SQL AUTO-INCREMENT

Very often we would like the value of the primary key field to be created automatically every time a new record is inserted.

5.7.1 Syntax for MySQL

The following SQL statement defines the "ID" column to be an auto-increment primary key field in the "Persons" table:

```
CREATE TABLE Persons
(
ID int NOT NULL AUTO_INCREMENT,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
PRIMARY KEY (ID)
)
```

MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature.

By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:

```
ALTER TABLE Persons AUTO_INCREMENT=100
```

To insert a new record into the "Persons" table, we will NOT have to specify a value for the "ID" column (a unique value will be added automatically):

```
INSERT INTO Persons (FirstName,LastName)
VALUES ('Lars','Monsen')
```

The SQL statement above would insert a new record into the "Persons" table. The "ID" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

5.7.2 Syntax for SQL Server

The following SQL statement defines the "ID" column to be an auto-increment primary key field in the "Persons" table:

```
CREATE TABLE Persons
(
ID int IDENTITY(1,1) PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

The MS SQL Server uses the IDENTITY keyword to perform an auto-increment feature.

In the example above, the starting value for IDENTITY is 1, and it will increment by 1 for each new record.

Tip: To specify that the "ID" column should start at value 10 and increment by 5, change it to IDENTITY(10,5).

To insert a new record into the "Persons" table, we will NOT have to specify a value for the "ID" column (a unique value will be added automatically):

```
INSERT INTO Persons (FirstName,LastName)
VALUES ('Lars','Monsen')
```

The SQL statement above would insert a new record into the "Persons" table. The "ID" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

5.7.3 Syntax for Microsoft Access

The following SQL statement defines the "ID" column to be an auto-increment primary key field in the "Persons" table:

```
CREATE TABLE Persons
(
ID Integer PRIMARY KEY AUTOINCREMENT,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

The MS Access uses the AUTOINCREMENT keyword to perform an auto-increment feature.

By default, the starting value for AUTOINCREMENT is 1, and it will increment by 1 for each new record.

Tip: To specify that the "ID" column should start at value 10 and increment by 5, change the autoincrement to AUTOINCREMENT(10,5).

To insert a new record into the "Persons" table, we will NOT have to specify a value for the "ID" column (a unique value will be added automatically):

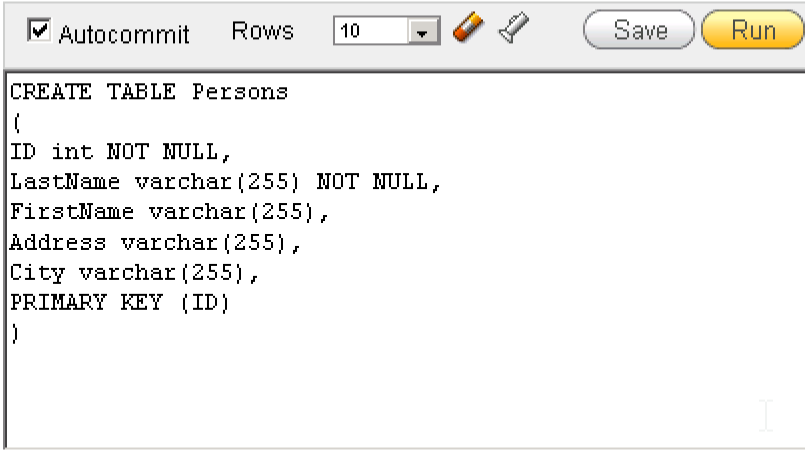
```
INSERT INTO Persons (FirstName,LastName)
VALUES ('Lars','Monsen')
```

The SQL statement above would insert a new record into the "Persons" table. The "P_Id" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

5.7.4 Syntax for Oracle (with simple ID)

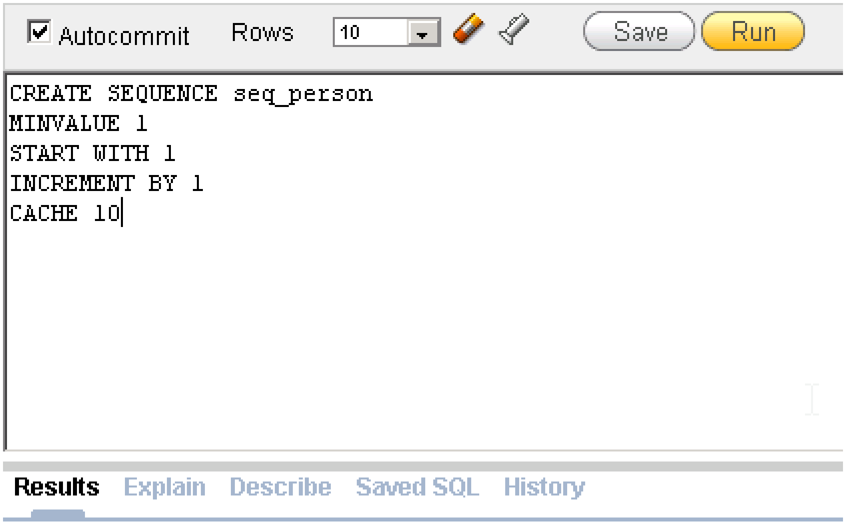
In Oracle the code is a little bit more tricky.

First we create this basic table:



You will have to create an auto-increment field with the sequence object (this object generates a number sequence).

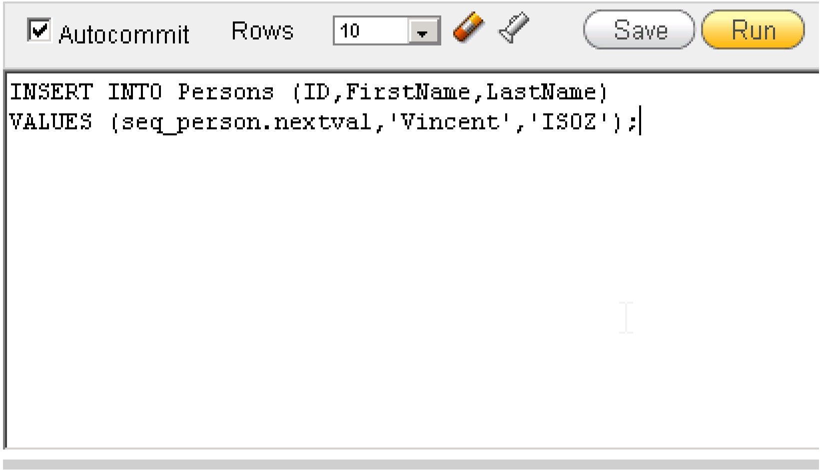
Use the following CREATE SEQUENCE syntax:




```
Sequence created.
```

The code above creates a sequence object called seq_person, that starts with 1 and will increment by 1. It will also cache up to 10 values for performance. The cache option specifies how many sequence values will be stored in memory for faster access.

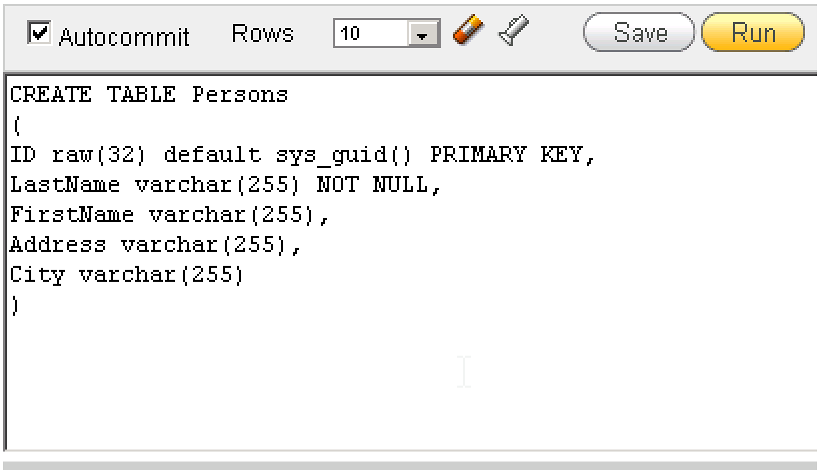
To insert a new record into the "Persons" table, we will have to use the nextval function (this function retrieves the next value from seq_person sequence) :



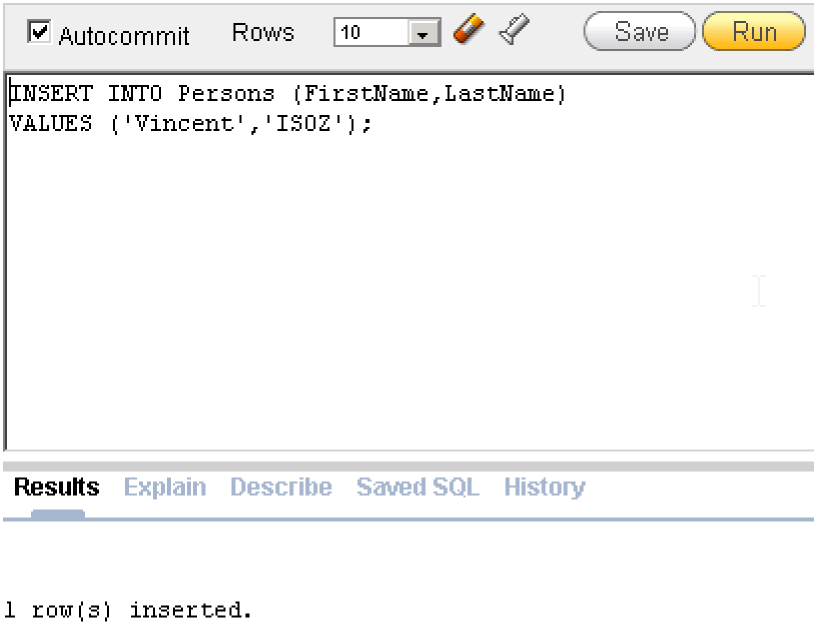
The SQL statement above would insert a new record into the "Persons" table. The "ID" column would be assigned the next number from the seq_person sequence. The "FirstName" column would be set to "Vincent" and the "LastName" column would be set to "ISOZ".

5.7.5 Syntax for Oracle (with GUID)

Using a GUID instead of a simple id auto-increment has some pros and cons (see *Database Modeling Course*). Then here we will focus on how to create such a thing in Oracle:



Then if you insert a new row:


Autocommit
Rows
10
Save
Run
INSERT INTO Persons (FirstName,LastName)
VALUES ('Vincent','ISOZ');
Results
Explain
Describe
Saved SQL
History
1 row(s) inserted.

You will get:

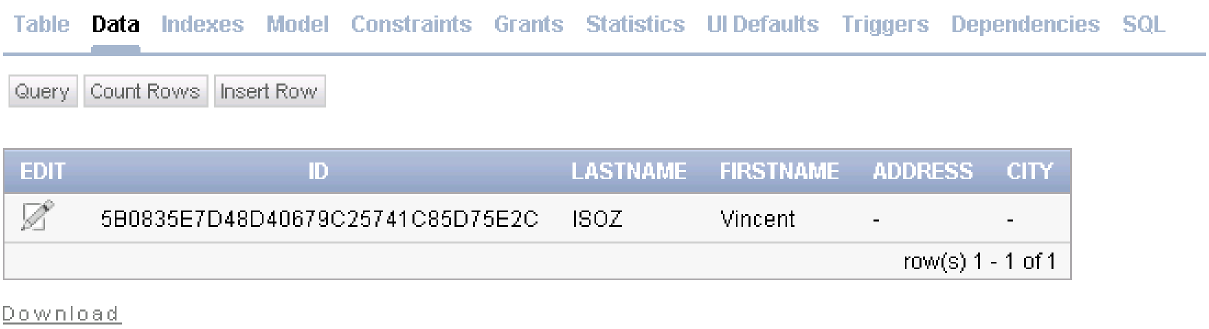


Table
Data
Indexes
Model
Constraints
Grants
Statistics
UI Defaults
Triggers
Dependencies
SQL
Query
Count Rows
Insert Row
EDIT
ID
LASTNAME
FIRSTNAME
ADDRESS
CITY
5B0835E7D48D40679C25741C85D75E2C
ISOZ
Vincent
-
-
row(s) 1 - 1 of 1
Download

6 SQL VIEWS

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

If a view contains the primary key and all others NOT NULL columns, the view can be used to insert datas or even override the original table constraints (by adding complementary constraints to the view). Here we will focus only on basic **read-only views** because this is the most common case for end-users (and we have only one week to study SQL...).

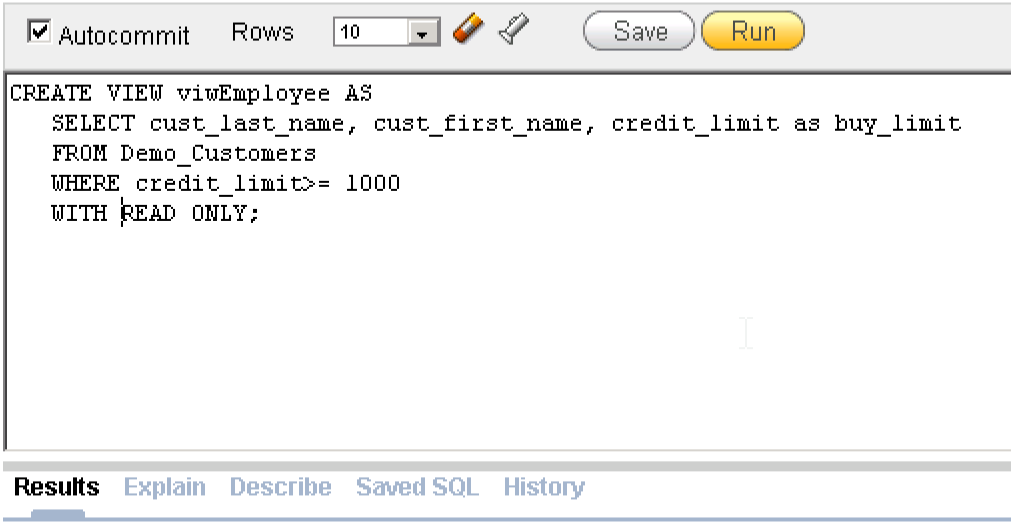
6.1 SQL CREATE VIEW Syntax

The general syntax is:

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

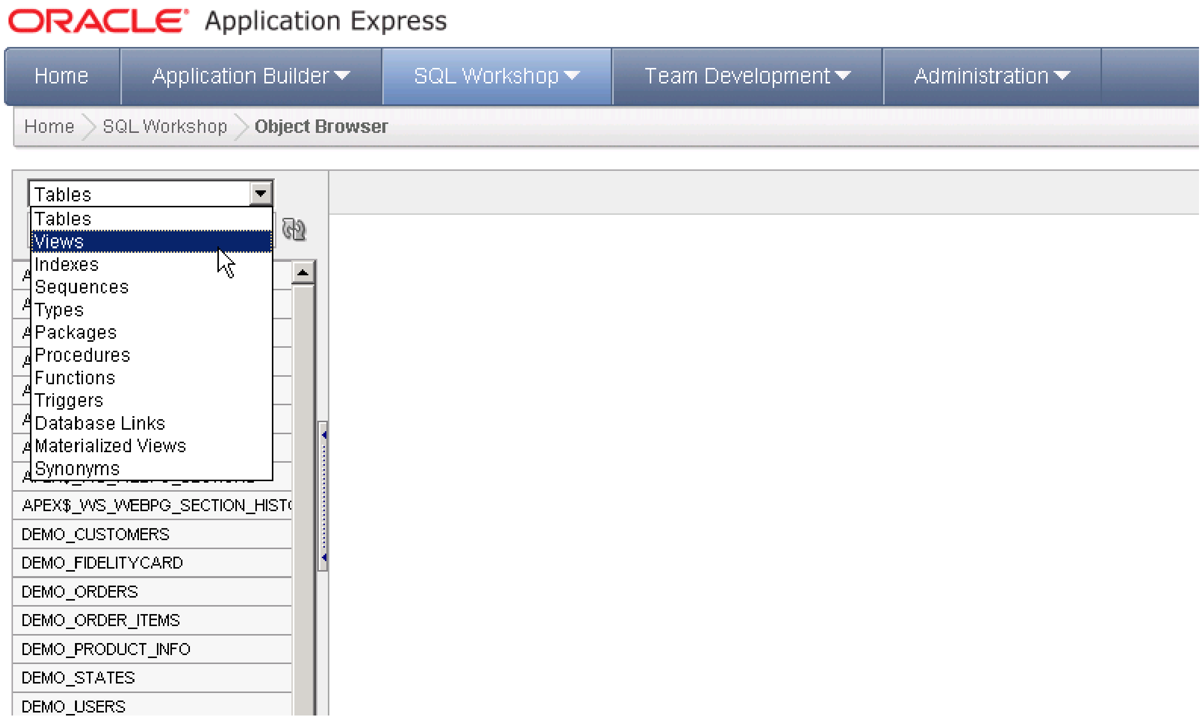
Note: A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

Also we begin with an example:

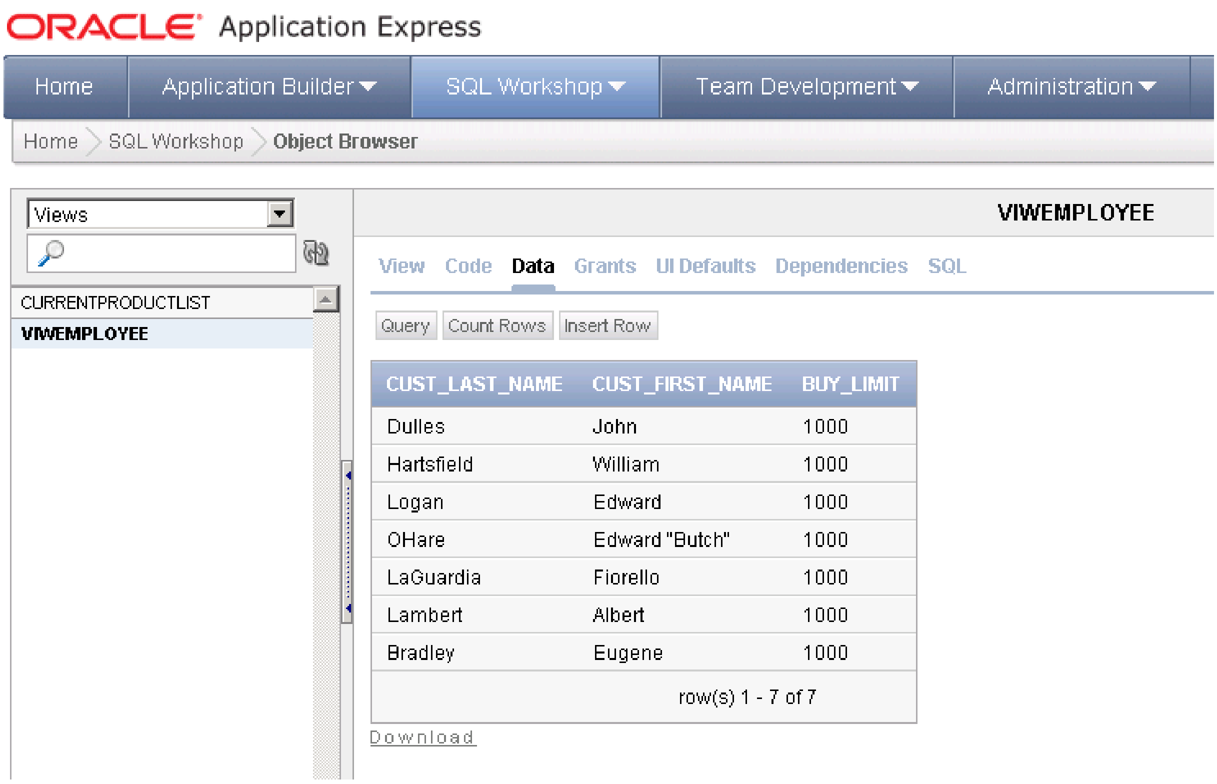



```
View created.
```

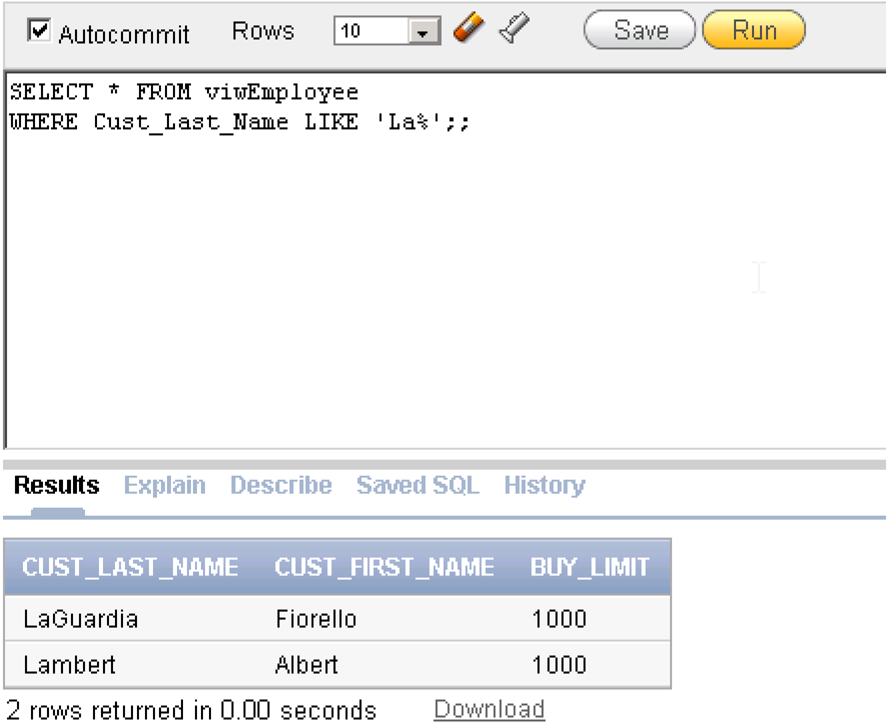
You can check that the view exists:



And then you will see the view:

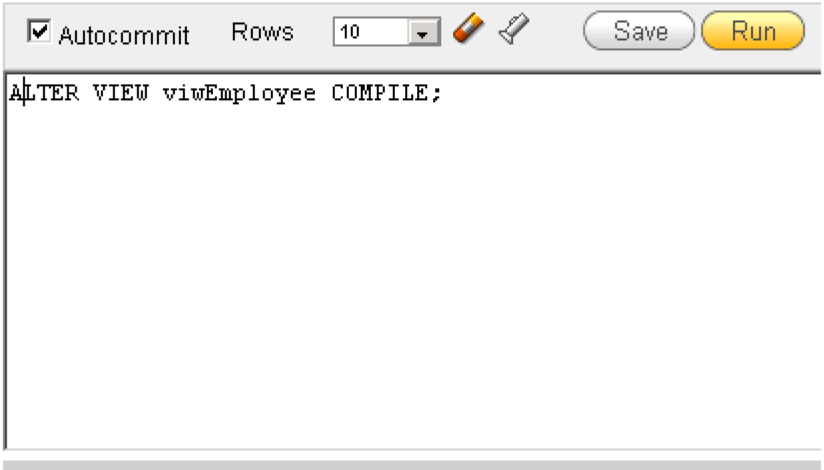


You can also query the view:

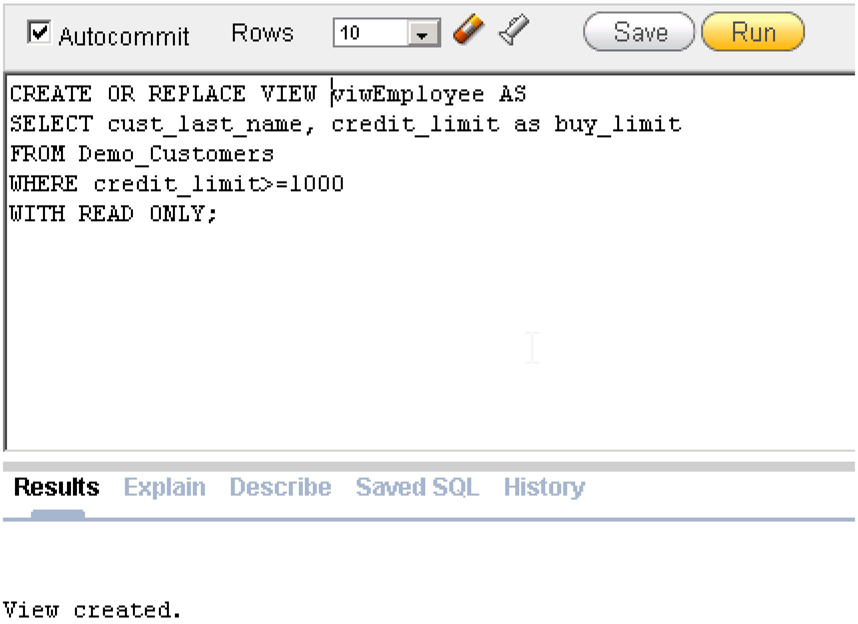

Autocommit
Rows
10
Save
Run
SELECT * FROM viwEmployee
WHERE Cust_Last_Name LIKE 'La%';;
Results
Explain
Describe
Saved SQL
History
CUST_LAST_NAME CUST_FIRST_NAME BUY_LIMIT
LaGuardia Fiorello 1000
Lambert Albert 1000
2 rows returned in 0.00 seconds
Download

6.2 SQL ALTER VIEW

If you change the structure of the table the view will not work anymore. Then you have to compile it:



No, you can't ALTER VIEW to add or remove columns! The syntax is the following (we don't want the cust_first_name column anymore):



6.3 SQL DROP VIEW

You can delete a view with the DROP VIEW command:

```
DROP VIEW view_name
```

7 SQL Functions

SQL has many built-in functions (almost ~150 for Oracle) for performing calculations on data. We will see here **only 19 functions** that have to be known by undergraduate students.

For more:

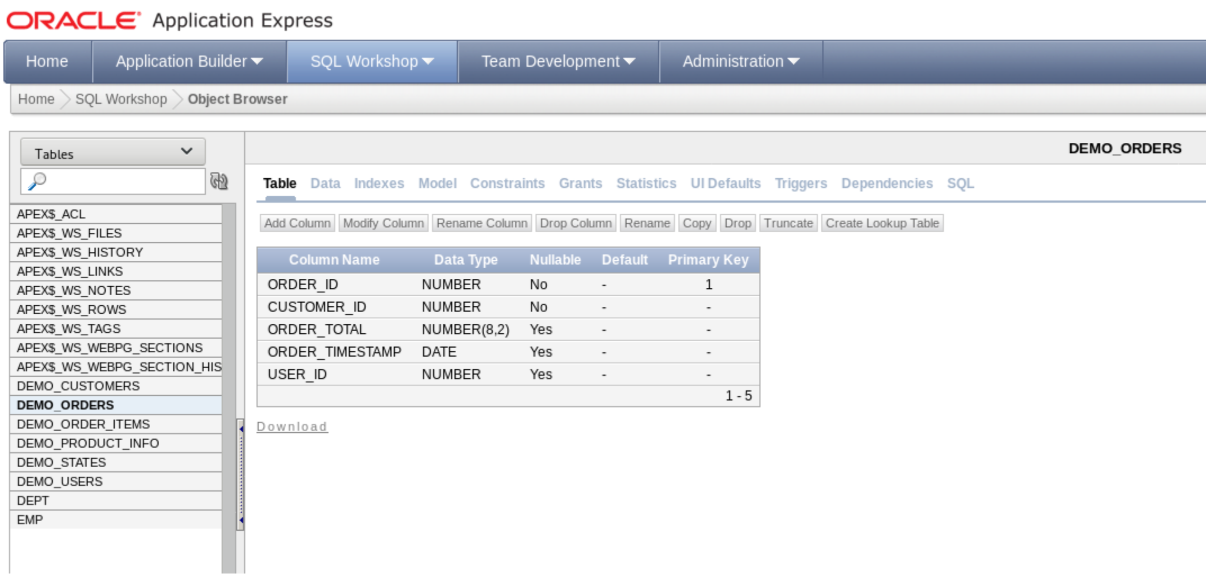
http://docs.oracle.com/cd/B19306_01/server.102/b14200/functions001.htm

http://www.techonthenet.com/oracle/functions/index.php

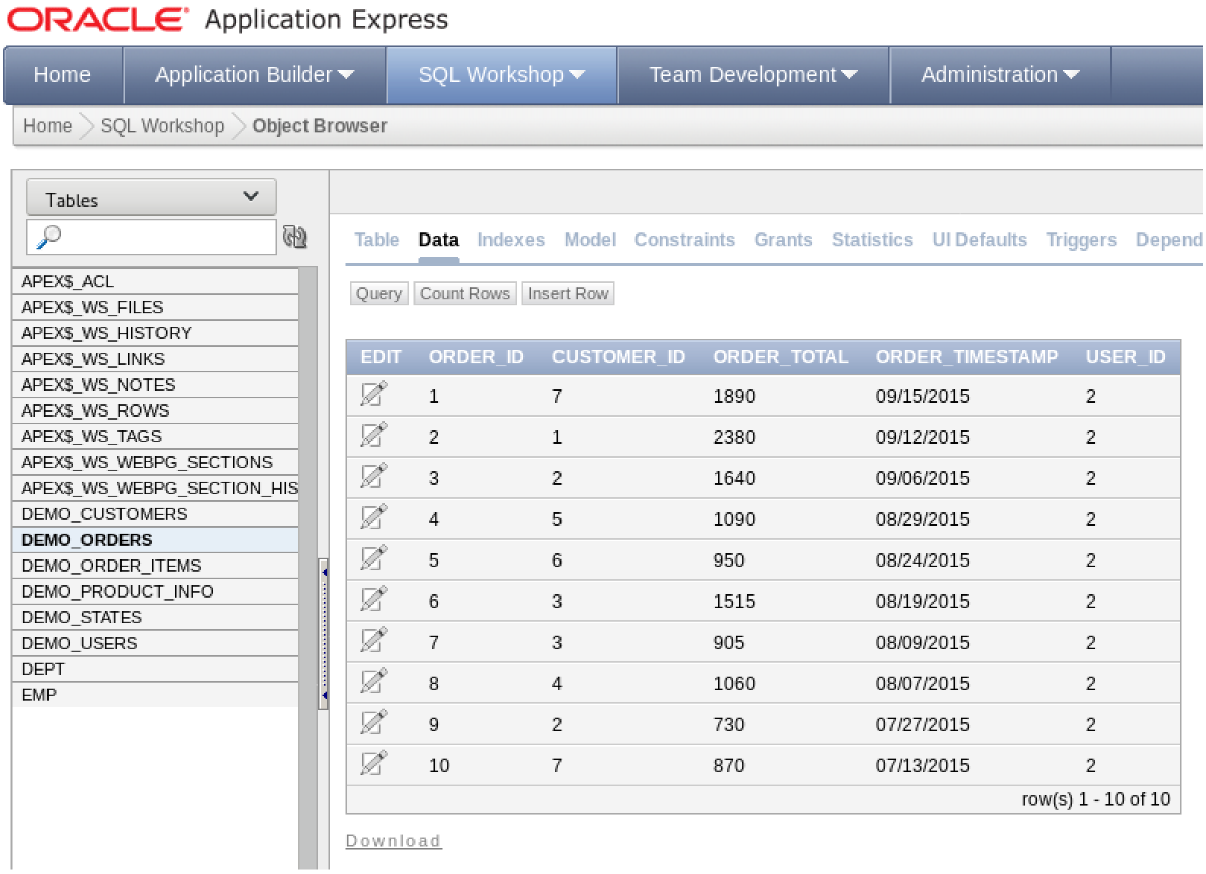
7.1.1 SQL CONVERSION function

The CAST() function converts a value (of any type) into a specified datatype.

For this let us consider the following demo table of Oracle Express:

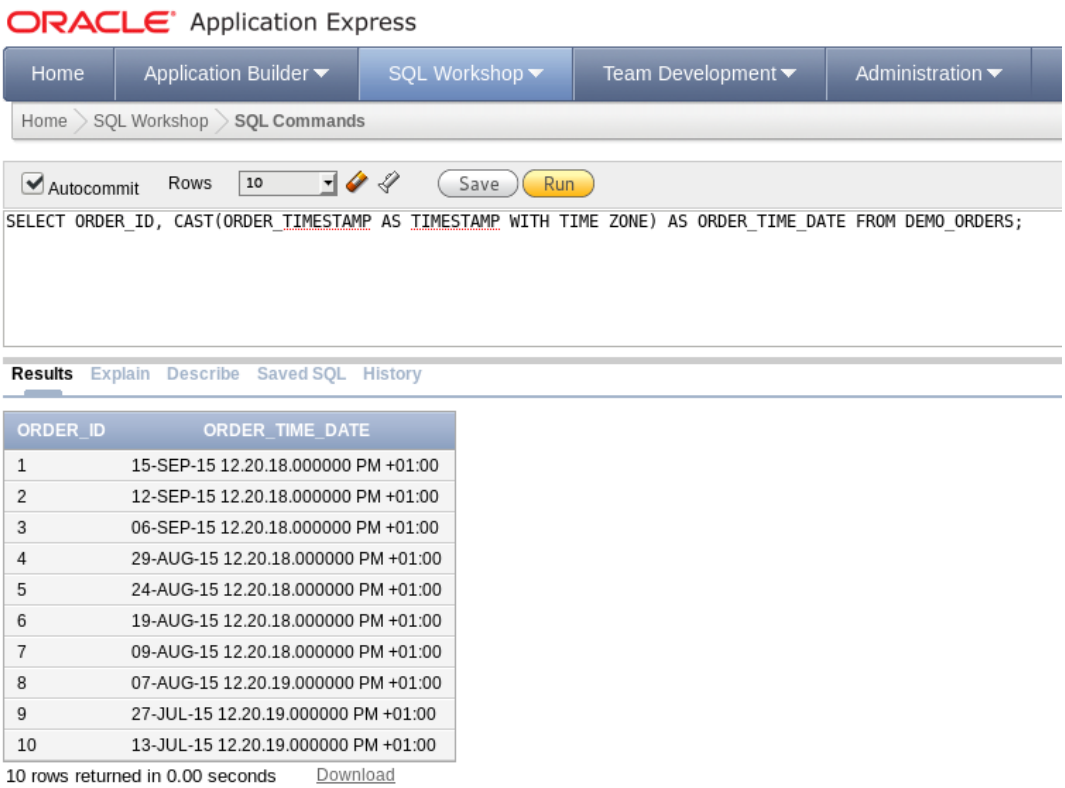


With the following content:



EDIT	ORDER_ID	CUSTOMER_ID	ORDER_TOTAL	ORDER_TIMESTAMP	USER_ID
	1	7	1890	09/15/2015	2
	2	1	2380	09/12/2015	2
	3	2	1640	09/06/2015	2
	4	5	1090	08/29/2015	2
	5	6	950	08/24/2015	2
	6	3	1515	08/19/2015	2
	7	3	905	08/09/2015	2
	8	4	1060	08/07/2015	2
	9	2	730	07/27/2015	2
	10	7	870	07/13/2015	2

And now let us see a typical example of the CAST( ) function:

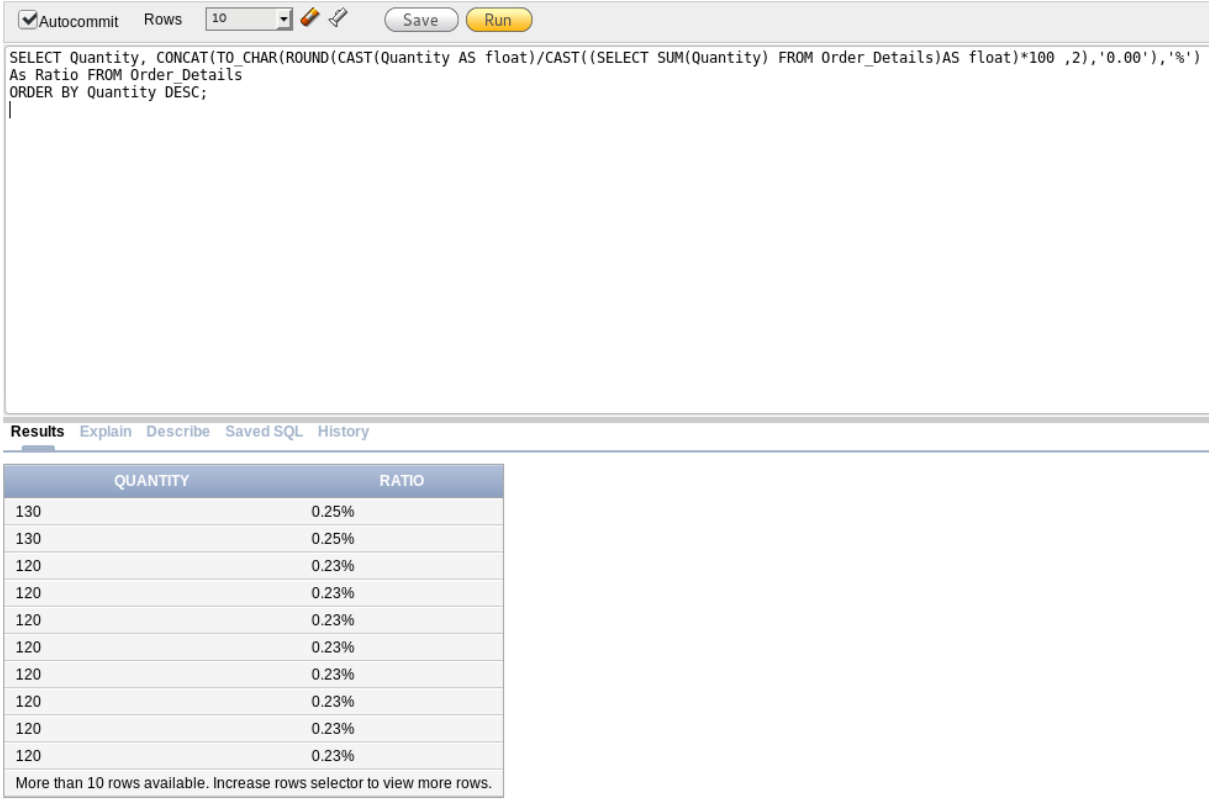



```sql
SELECT ORDER_ID, CAST(ORDER_TIMESTAMP AS TIMESTAMP WITH TIME ZONE) AS ORDER_TIME_DATE FROM DEMO_ORDERS;
```

ORDER_ID	ORDER_TIME_DATE
1	15-SEP-15 12.20.18.000000 PM +01:00
2	12-SEP-15 12.20.18.000000 PM +01:00
3	06-SEP-15 12.20.18.000000 PM +01:00
4	29-AUG-15 12.20.18.000000 PM +01:00
5	24-AUG-15 12.20.18.000000 PM +01:00
6	19-AUG-15 12.20.18.000000 PM +01:00
7	09-AUG-15 12.20.18.000000 PM +01:00
8	07-AUG-15 12.20.19.000000 PM +01:00
9	27-JUL-15 12.20.19.000000 PM +01:00
10	13-JUL-15 12.20.19.000000 PM +01:00

Obviously we can also convert to INT (integer), to VARCHAR(…) (string) and so on… corresponding to all standard column types in Oracle.

An important example is the following of CAST is also the ratio of two integers. Some Database will not return a result for the ration of two integers. This is why by security we will always write something like:


Autocommit Rows 10 Save Run

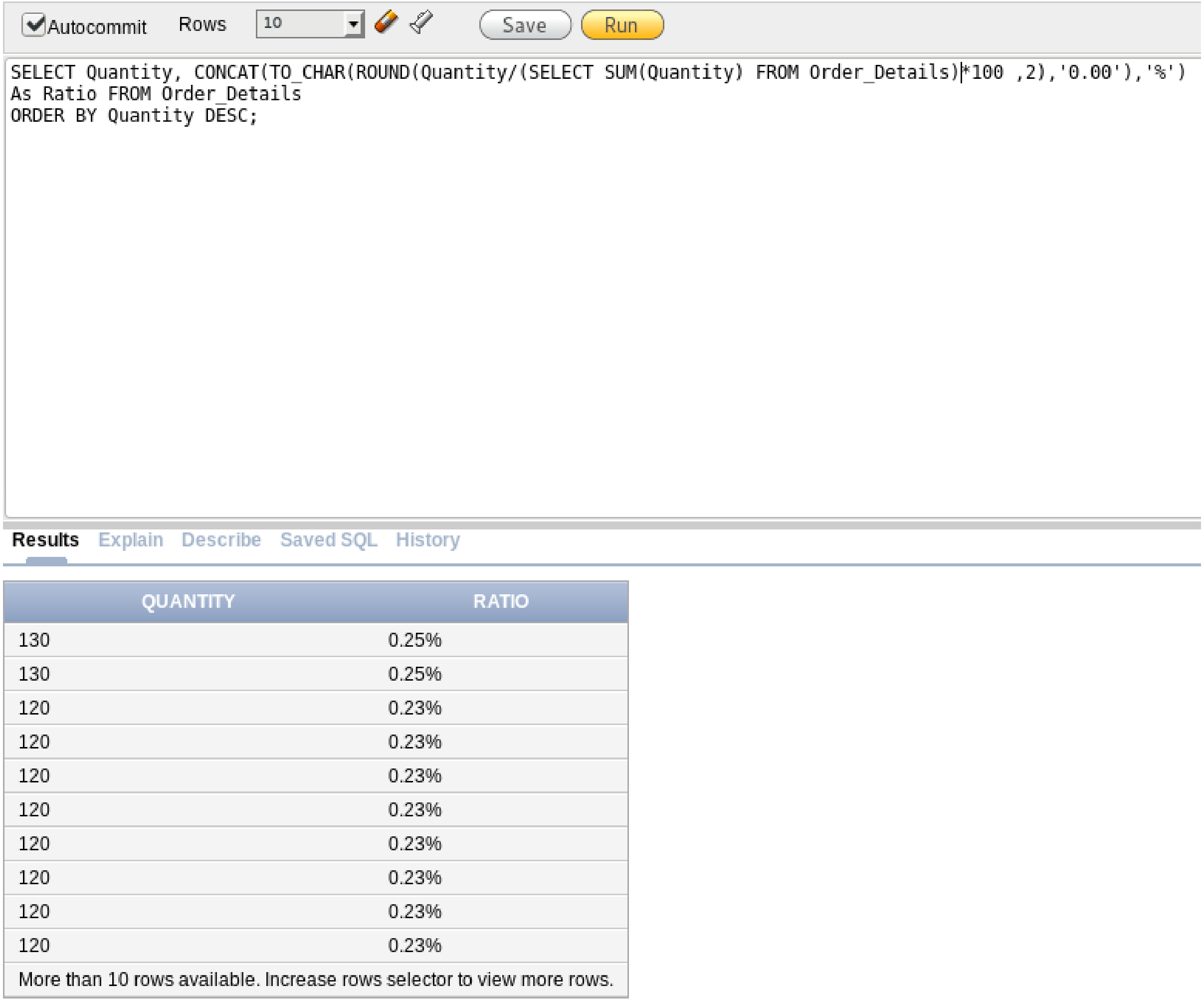
```
SELECT Quantity, CONCAT(TO_CHAR(ROUND(CAST(Quantity AS float)/CAST((SELECT SUM(Quantity) FROM Order_Details)AS float)*100 ,2),'0.00'),'%')
As Ratio FROM Order_Details
ORDER BY Quantity DESC;
```

Results Explain Describe Saved SQL History

QUANTITY	RATIO
130	0.25%
130	0.25%
120	0.23%
120	0.23%
120	0.23%
120	0.23%
120	0.23%
120	0.23%
120	0.23%
120	0.23%

More than 10 rows available. Increase rows selector to view more rows.

Instead of:



QUANTITY	RATIO
130	0.25%
130	0.25%
120	0.23%
120	0.23%
120	0.23%
120	0.23%
120	0.23%
120	0.23%
120	0.23%
120	0.23%

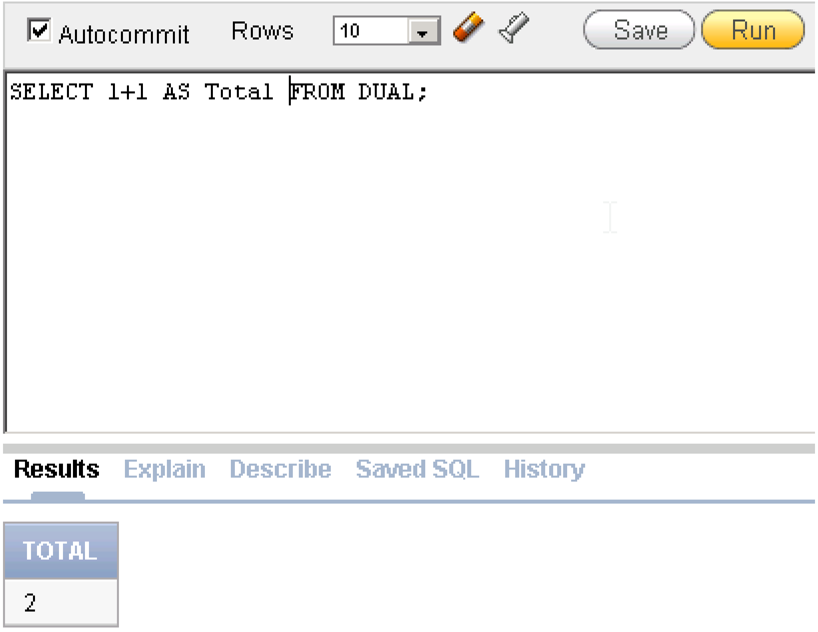
7.1.2 SQL AGGREGATE functions

To study the family of AGGREGATE functions we will use a mix of the W3 School website and Oracle!

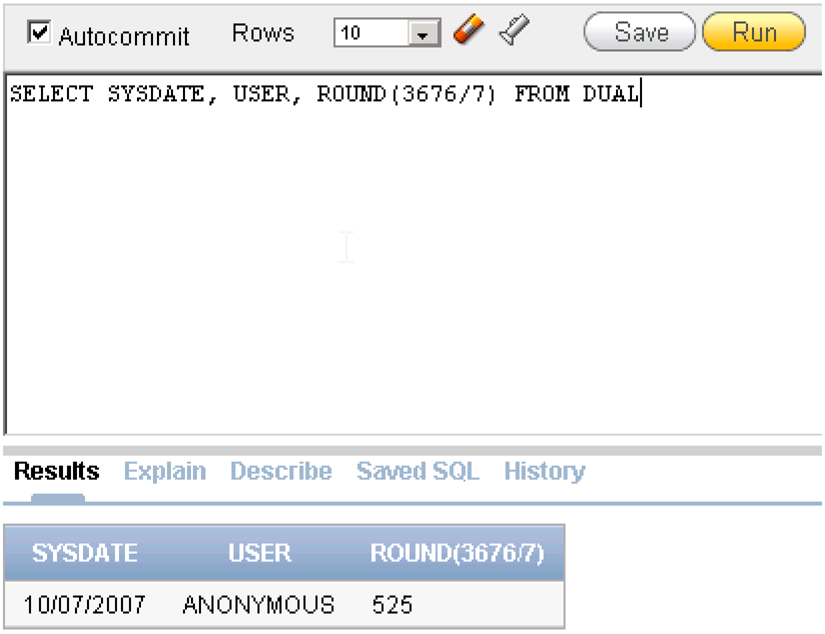
7.1.2.1 Dual Table

But first let us introduce the DUAL table, that is a special one-column table present by default in all Oracle database installations. It is suitable for use in testing simple functions.

For example:



or for fun:



etc... OK now let's go with aggregation functions!

7.1.2.2 SQL GROUP BY function

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

SQL GROUP BY Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;
```

Always put the WHERE statement before the GROUP BY otherwise you may filter on a column that doesn't exist anymore because of the grouping!

Below is a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2
...				

And a selection from the "Shippers" table:

ShipperID	ShipperName	Phone
1	Speedy Express	(503) 555-9831
2	United Package	(503) 555-3199
3	Federal Shipping	(503) 555-9931
...		

And a selection from the "Employees" table:

EmployeeID	LastName	FirstName	BirthDate	Photo	Notes
1	Davolio	Nancy	1968-12-08	EmpID1.pic	Education includes a BA....
2	Fuller	Andrew	1952-02-19	EmpID2.pic	Andrew received his BTS....
3	Leverling	Janet	1963-08-30	EmpID3.pic	Janet has a BS degree....
...					

Now we want to find the number of orders sent by each shipper.

```
SELECT Shippers.ShipperName,COUNT(Orders.OrderID) AS NumberOfOrders FROM
Orders
```

```
LEFT JOIN Shippers
ON Orders.ShipperID=Shippers.ShipperID
GROUP BY ShipperName;
```

The result will be:

ShipperName	NumberOfOrders
Federal Shipping	68
Speedy Express	54
United Package	74

We can also use the GROUP BY statement on more than one column, like this:

```
SELECT Shippers.ShipperName, Employees.LastName,
COUNT(Orders.OrderID) AS NumberOfOrders
FROM ((Orders
INNER JOIN Shippers
ON Orders.ShipperID=Shippers.ShipperID)
INNER JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID)
GROUP BY ShipperName,LastName
ORDER BY 3;
```

The result will be:

ShipperName	LastName	NumberOfOrders
Speedy Express	Dodsworth	2
Federal Shipping	Suyama	3
Speedy Express	Buchanan	3
Speedy Express	King	3
United Package	Buchanan	3
Federal Shipping	Dodsworth	4
Federal Shipping	Fuller	4
United Package	King	4
Federal Shipping	Buchanan	5
Speedy Express	Callahan	5
Federal Shipping	King	7
Speedy Express	Fuller	7
Speedy Express	Leverling	7
Speedy Express	Suyama	7

Speedy Express	Davolio	8
....		

7.1.2.3 SQL GROUP BY with HAVING function

The HAVING clause was added to SQL **because the WHERE keyword could not be used with aggregate functions**.

SQL HAVING Syntax:

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value;
```

Below is a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2
...				

And a selection from the "Employees" table:

EmployeeID	LastName	FirstName	BirthDate	Photo	Notes
1	Davolio	Nancy	1968-12-08	EmpID1.pic	Education includes a BA....
2	Fuller	Andrew	1952-02-19	EmpID2.pic	Andrew received his BTS....
3	Leverling	Janet	1963-08-30	EmpID3.pic	Janet has a BS degree....
...					

The following SQL statement finds if any of the employees has registered more than 10 orders:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders FROM
(Orders
INNER JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

The result will be:

LastName	NumberOfOrders
Buchanan	11
Callahan	27
Davolio	29

Fuller	20
King	14
Leverling	31
Peacock	40
Suyama	18

Now we want to find if the employees "Davolio" or "Fuller" have more than 25 orders

7.1.2.4 Mixing HAVING and WHERE

We can add an ordinary WHERE clause to the SQL statement.

Example using the same table as before:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders FROM
Orders
INNER JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID
WHERE LastName='Davolio' OR LastName='Fuller'
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

But this is equivalent to:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders FROM
Orders
INNER JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25  AND (LastName='Davolio' OR
LastName='Fuller');
```

and don't forget the parenthesis after the AND logical operator otherwise the result won't be the same.

The result will be:

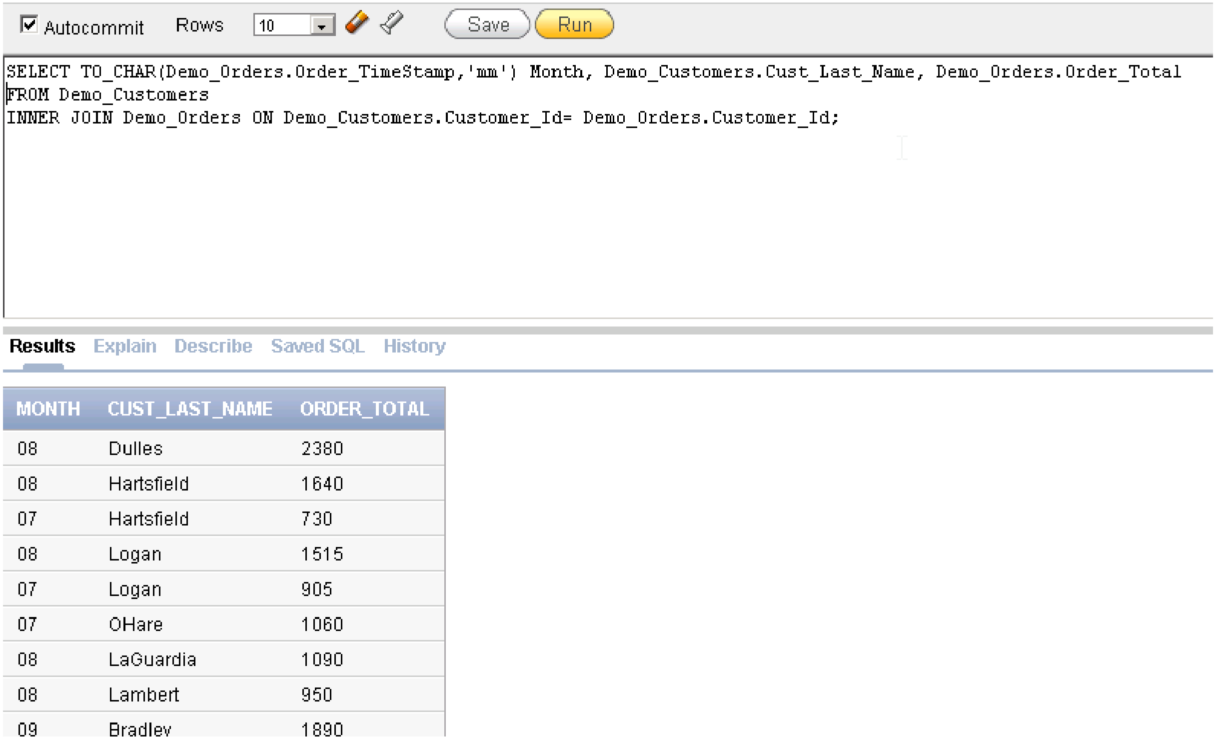
tName	NumberOfOrders
Davolio	29

7.1.2.5 SQL GROUP BY ROLLUP (crosstab queries)

Grouping and in particular gathering aggregates across groups often brings confusion to many practitioners. This does not need to be the case if approached from a systematic fashion. This article will approach gathering aggregates from a simple GROUP BY operation and then extend into Oracle's more higher-level grouping operations. In particular the ROLLUP operation which allows us to group and aggregate at different levels in a collection of similar rows.

To see how the GROUP BY ROLLUP works, we will focus once again on Oracle.

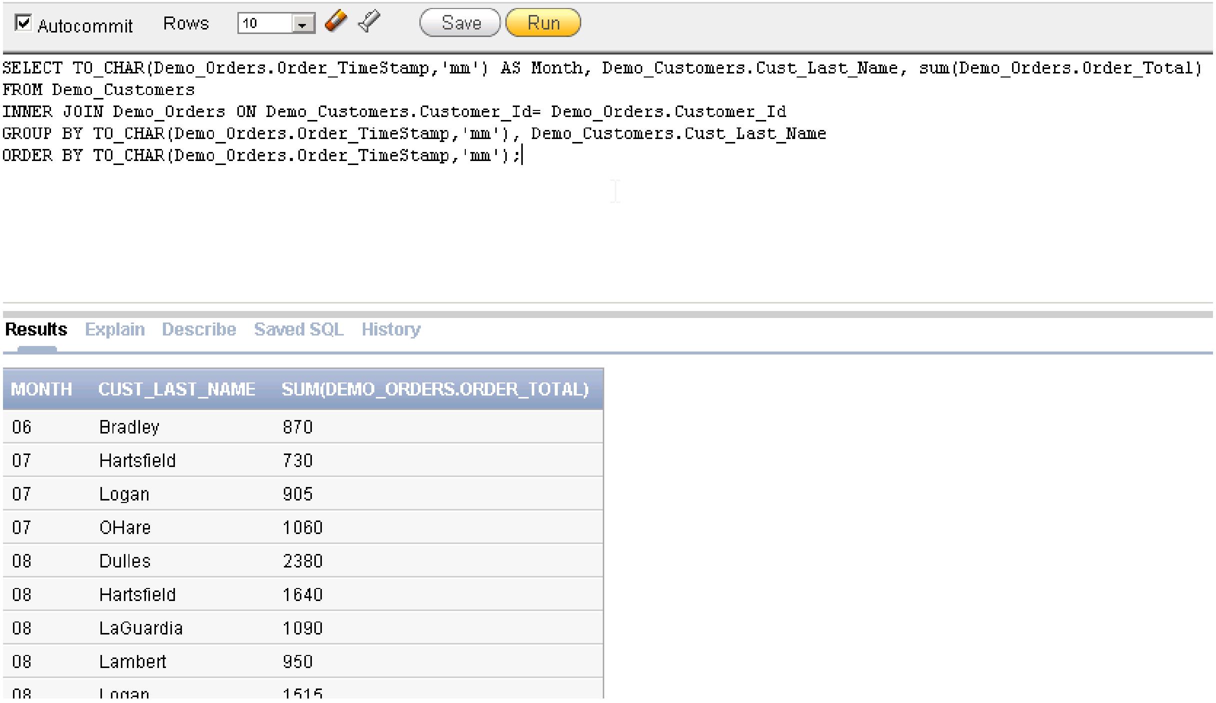
First, we make a basic simple query:




```
SELECT TO_CHAR(Demo_Orders.Order_TimeStamp,'mm') Month, Demo_Customers.Cust_Last_Name, Demo_Orders.Order_Total
FROM Demo_Customers
INNER JOIN Demo_Orders ON Demo_Customers.Customer_Id= Demo_Orders.Customer_Id;
```

MONTH	CUST_LAST_NAME	ORDER_TOTAL
08	Dulles	2380
08	Hartsfield	1640
07	Hartsfield	730
08	Logan	1515
07	Logan	905
07	OHare	1060
08	LaGuardia	1090
08	Lambert	950
09	Bradley	1890

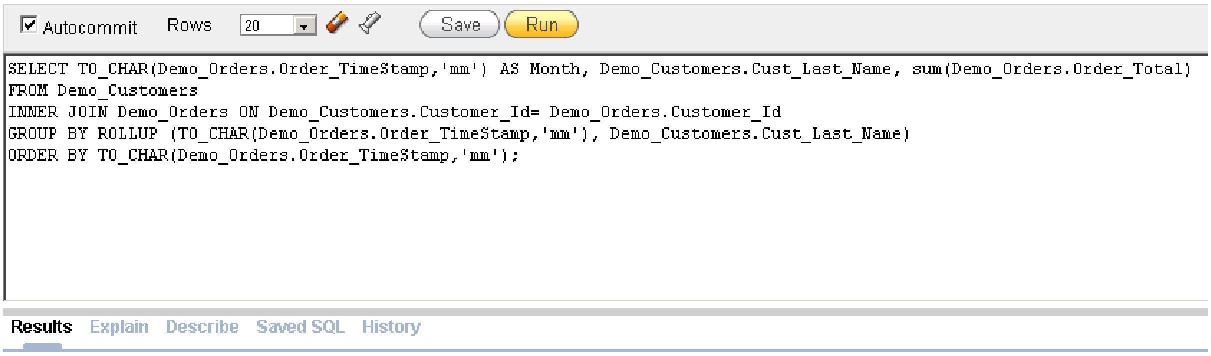
We complexify a little bit this query by adding a GROUP BY and ORDER BY statement and a sum( ) on the order total:




```sql
SELECT TO_CHAR(Demo_Orders.Order_TimeStamp,'mm') AS Month, Demo_Customers.Cust_Last_Name, sum(Demo_Orders.Order_Total)
FROM Demo_Customers
INNER JOIN Demo_Orders ON Demo_Customers.Customer_Id= Demo_Orders.Customer_Id
GROUP BY TO_CHAR(Demo_Orders.Order_TimeStamp,'mm'), Demo_Customers.Cust_Last_Name
ORDER BY TO_CHAR(Demo_Orders.Order_TimeStamp,'mm');
```

MONTH	CUST_LAST_NAME	SUM(DEMO_ORDERS.ORDER_TOTAL)
06	Bradley	870
07	Hartsfield	730
07	Logan	905
07	OHare	1060
08	Dulles	2380
08	Hartsfield	1640
08	LaGuardia	1090
08	Lambert	950
08	Logan	1515

And now comes the ROLLUP:




```sql
SELECT TO_CHAR(Demo_Orders.Order_TimeStamp,'mm') AS Month, Demo_Customers.Cust_Last_Name, sum(Demo_Orders.Order_Total)
FROM Demo_Customers
INNER JOIN Demo_Orders ON Demo_Customers.Customer_Id= Demo_Orders.Customer_Id
GROUP BY ROLLUP (TO_CHAR(Demo_Orders.Order_TimeStamp,'mm'), Demo_Customers.Cust_Last_Name)
ORDER BY TO_CHAR(Demo_Orders.Order_TimeStamp,'mm');
```

Results Explain Describe Saved SQL History

MONTH	CUST_LAST_NAME	SUM(DEMO_ORDERS.ORDER_TOTAL)
06	Bradley	870
06	-	870
07	Hartsfield	730
07	Logan	905
07	OHare	1060
07	-	2695
08	Dulles	2380
08	Hartsfield	1640
08	LaGuardia	1090
08	Lambert	950
08	Logan	1515
08	-	7575
09	Bradley	1890
09	-	1890
-	-	13030

As you can see this do the same as a GROUP BY but adds sub-totals rows and at the end at grand-total! This is especially useful for invoices automation purposes.

And if you have the time you can mix all the stuff study we saw until now:

```
SELECT state,
       round( sum( mens ), 2 ) "Mens",
       round( sum( womens ), 0 ) "Womens",
       round( sum( accessories ), 0 ) "Accessories"
FROM   ( SELECT demo_customers.cust_state state,
                CASE
                   WHEN demo_product_info.category = 'Mens'
                   THEN
                      demo_order_items.quantity *
demo_order_items.unit_price
                   ELSE
                      0
                END
                   mens,
                CASE
                   WHEN demo_product_info.category = 'Womens'
                   THEN
                      demo_order_items.quantity *
demo_order_items.unit_price
                   ELSE
                      0
                END
                   womens,
                CASE
                   WHEN demo_product_info.category = 'Accessories'
                   THEN
```

```
                      demo_order_items.quantity *
demo_order_items.unit_price
                   ELSE
                      0
                END
                   accessories
         FROM   demo_order_items,
                demo_product_info,
                demo_customers,
                demo_orders
         WHERE  demo_order_items.product_id = demo_product_info.product_id
                AND demo_order_items.order_id = demo_orders.order_id
                AND demo_orders.customer_id = demo_customers.customer_id )
GROUP BY ROLLUP( state )
```

You will get:

STATE	Mens	Womens	Accessories
CT	2760	0	0
GA	0	1520	850
IL	940	120	0
MA	1040	640	740
MO	610	340	0
NY	220	240	630
VA	1060	660	660
-	6630	3520	2880

7.1.2.6 SQL GROUP BY CUBE (crosstab queries)

The GROUP BY CUBE can be seen as an extension of the GROUP BY ROLLUP because it adds complementary subtotal at the end of the returned table:

Autocommit Rows 30 Save Run

```
SELECT TO_CHAR(Demo_Orders.Order_TimeStamp,'mm') AS Month, Demo_Customers.Cust_Last_Name, sum(Demo_Orders.Order_Total)
FROM Demo_Customers
INNER JOIN Demo_Orders ON Demo_Customers.Customer_Id= Demo_Orders.Customer_Id
GROUP BY CUBE(TO_CHAR(Demo_Orders.Order_TimeStamp,'mm'), Demo_Customers.Cust_Last_Name)
ORDER BY TO_CHAR(Demo_Orders.Order_TimeStamp,'mm');
```

Results Explain Describe Saved SQL History

MONTH	CUST_LAST_NAME	SUM(DEMO_ORDERS.ORDER_TOTAL)
06	Bradley	870
06	-	870
07	Hartsfield	730
07	Logan	905
07	OHare	1060
07	-	2695
08	Dulles	2380
08	Hartsfield	1640
08	LaGuardia	1090
08	Lambert	950
08	Logan	1515
08	-	7575
09	Bradley	1890
09	-	1890
-	Bradley	2760
-	Dulles	2380
-	Hartsfield	2370
-	LaGuardia	1090
-	Lambert	950
-	Logan	2420
-	OHare	1060
-	-	13030

7.1.2.6.1 SQL GROUPING statement

To make this result more usable in some special case, some users add grouping:

```sql
SELECT TO_CHAR(Demo_Orders.Order_TimeStamp,'mm') AS Month, Demo_Customers.Cust_Last_Name, SUM(Demo_Orders.Order_Total),
  GROUPING(TO_CHAR(Demo_Orders.Order_TimeStamp,'mm')) AS f1g,
  GROUPING(Demo_Customers.Cust_Last_Name) AS f2g
FROM Demo_Customers
  INNER JOIN Demo_Orders ON Demo_Customers.Customer_Id=Demo_Orders.Customer_Id
GROUP BY CUBE (TO_CHAR(Demo_Orders.Order_TimeStamp,'mm'),Demo_Customers.Cust_Last_name)
ORDER BY TO_CHAR(Demo_Orders.Order_TimeStamp,'mm');
```

MONTH	CUST_LAST_NAME	SUM(DEMO_ORDERS.ORDER_TOTAL)	F1G	F2G
06	Bradley	870	0	0
06	-	870	0	1
07	Hartsfield	730	0	0
07	Logan	905	0	0
07	OHare	1060	0	0
07	-	2695	0	1
08	Dulles	2380	0	0
08	Hartsfield	1640	0	0
08	LaGuardia	1090	0	0
08	Lambert	950	0	0
08	Logan	1515	0	0
08	-	7575	0	1
09	Bradley	1890	0	0
09	-	1890	0	1
-	Bradley	2760	1	0
-	Dulles	2380	1	0
-	Hartsfield	2370	1	0
-	LaGuardia	1090	1	0
-	Lambert	950	1	0
-	Logan	2420	1	0
-	OHare	1060	1	0
-	-	13030	1	1

in this way it's easier use the result as a subquery.

7.1.2.6.2 SQL GROUPING_ID statement

Or as an easier alternative to GROUPING as seen above, you can use GROUPING_ID:

Autocommit Rows 30 Save Run

```
SELECT TO_CHAR(Demo_Orders.Order_TimeStamp,'mm') AS Month, Demo_Customers.Cust_Last_Name, SUM(Demo_Orders.Order_Total),
  GROUPING_ID(TO_CHAR(Demo_Orders.Order_TimeStamp,'mm'), Demo_Customers.Cust_Last_Name) AS grouping_id
FROM Demo_Customers
  INNER JOIN Demo_Orders ON Demo_Customers.Customer_Id=Demo_Orders.Customer_Id
GROUP BY CUBE (TO_CHAR(Demo_Orders.Order_TimeStamp,'mm'),Demo_Customers.Cust_Last_name)
ORDER BY TO_CHAR(Demo_Orders.Order_TimeStamp,'mm');
```

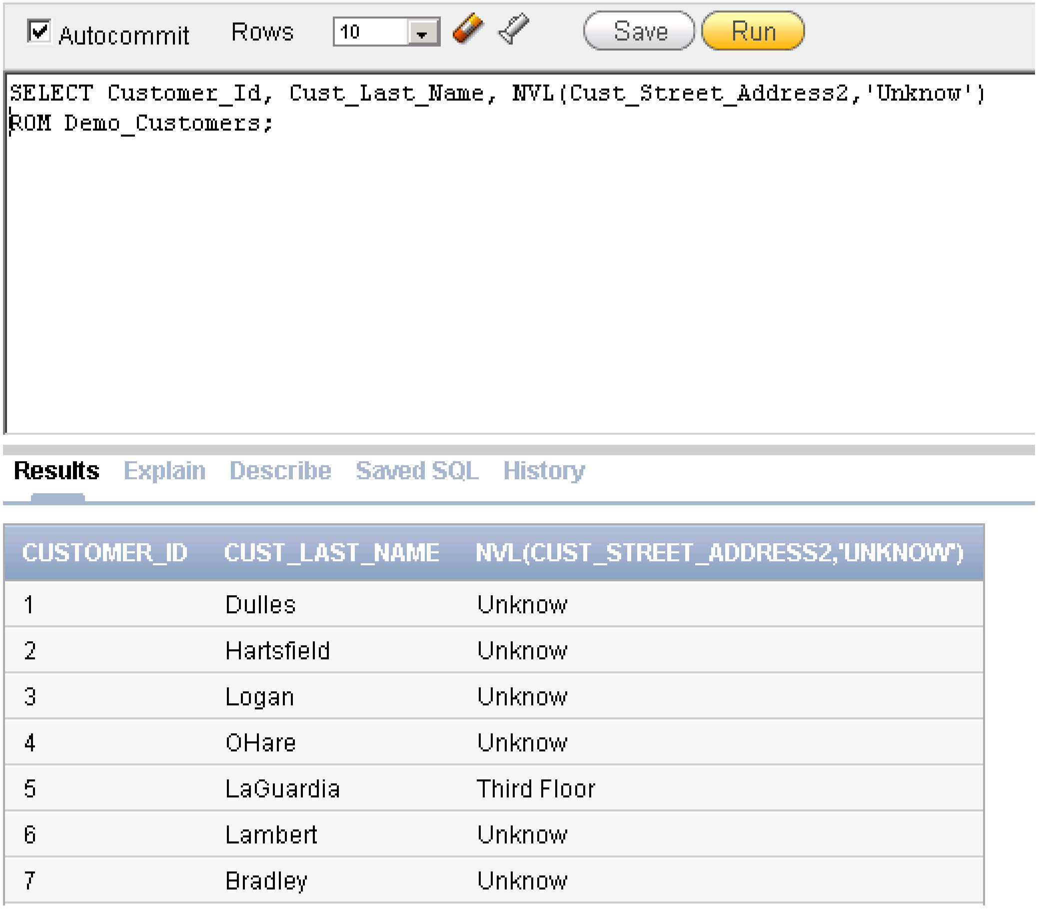
Results Explain Describe Saved SQL History

MONTH	CUST_LAST_NAME	SUM(DEMO_ORDERS.ORDER_TOTAL)	GROUPING_ID
06	Bradley	870	0
06	-	870	1
07	Hartsfield	730	0
07	Logan	905	0
07	OHare	1060	0
07	-	2695	1
08	Dulles	2380	0
08	Hartsfield	1640	0
08	LaGuardia	1090	0
08	Lambert	950	0
08	Logan	1515	0
08	-	7575	1
09	Bradley	1890	0
09	-	1890	1
-	Bradley	2760	2
-	Dulles	2380	2
-	Hartsfield	2370	2
-	LaGuardia	1090	2
-	Lambert	950	2
-	Logan	2420	2
-	OHare	1060	2
-	-	13030	3

7.1.3 SQL Null Management functions

7.1.3.1 SQL NVL

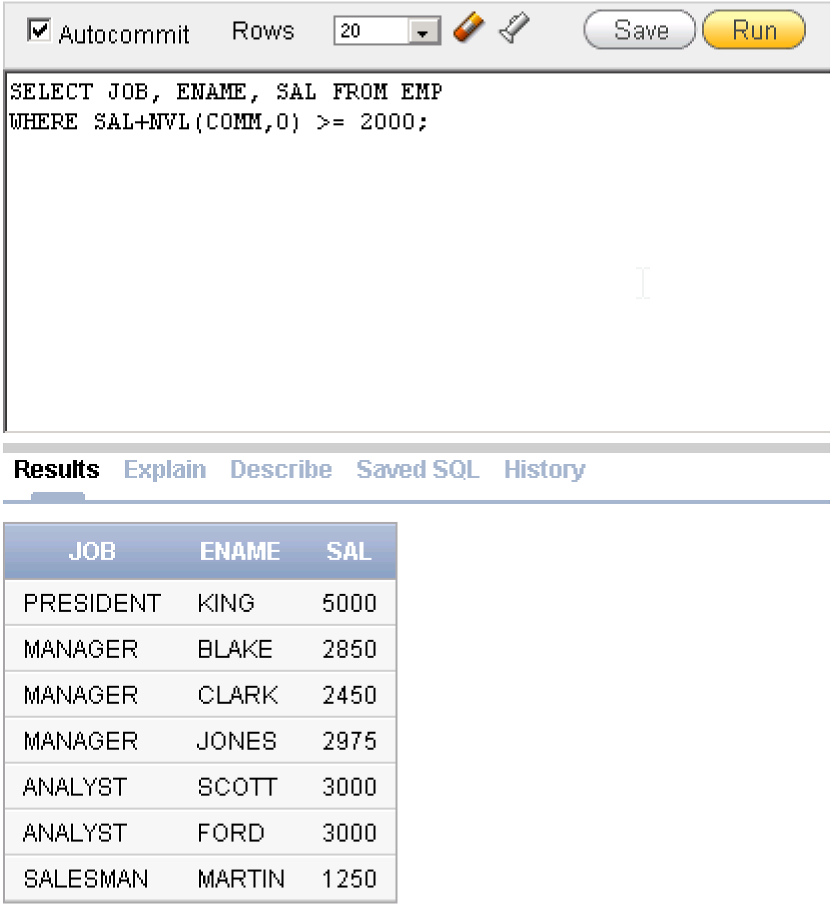
In Oracle/PLSQL, the NVL( ) function lets you substitute a value when a null value is encountered.



The same syntax on SQL Server will be:

```
SELECT Customer_Id, Cust_Last_Name, ISNULL(Cust_Street_Address2,'Unknow')
FROM Demo_Customers;
```

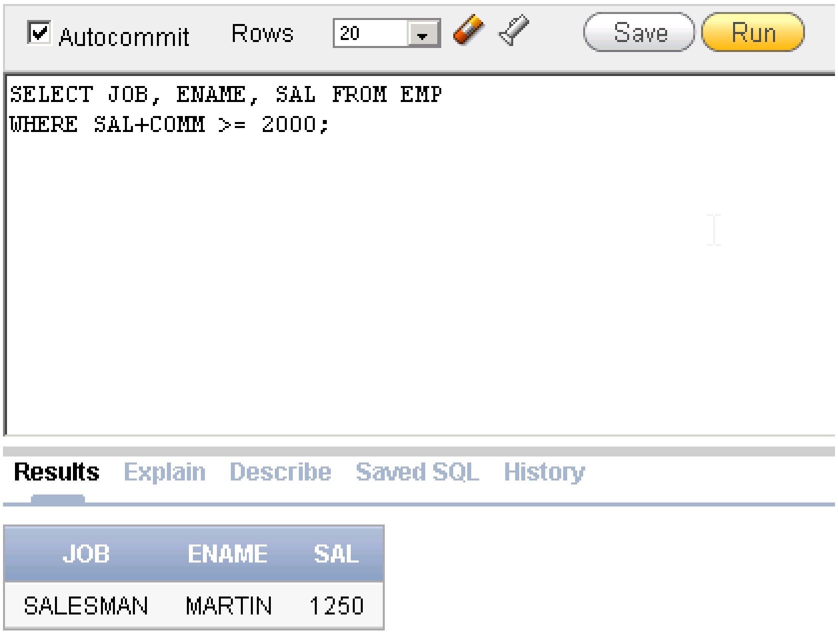
Another interesting example is the following:




```
SELECT JOB, ENAME, SAL FROM EMP
WHERE SAL+NVL(COMM,0) >= 2000;
```

JOB	ENAME	SAL
PRESIDENT	KING	5000
MANAGER	BLAKE	2850
MANAGER	CLARK	2450
MANAGER	JONES	2975
ANALYST	SCOTT	3000
ANALYST	FORD	3000
SALESMAN	MARTIN	1250

to compare with:



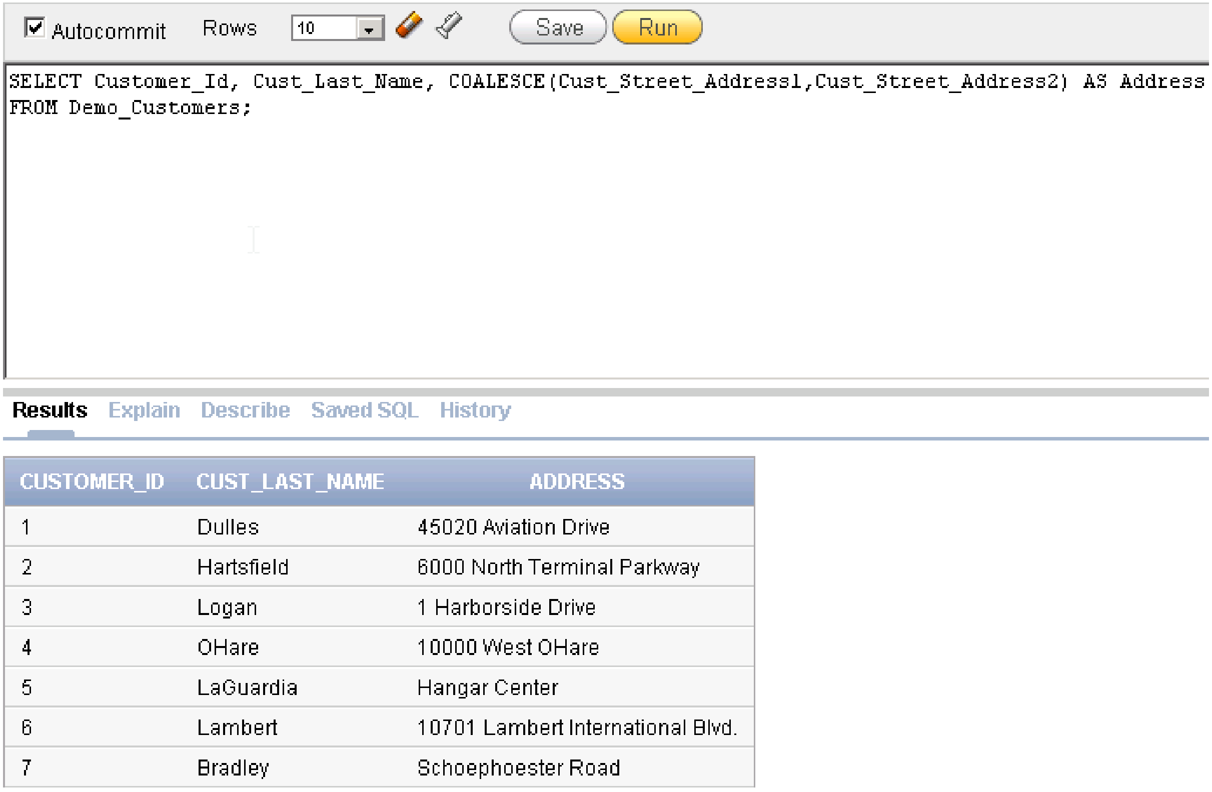

```
SELECT JOB, ENAME, SAL FROM EMP
WHERE SAL+COMM >= 2000;
```

JOB	ENAME	SAL
SALESMAN	MARTIN	1250

nice trap... this is why triple check is important when you manage billion dollars!

7.1.3.2 SQL COALESCE Function

In Oracle/PLSQL, the COALESCE( ) function returns the first non-null expression in the list. If all expressions evaluate to null, then the COALESCE function will return null.




```
SELECT Customer_Id, Cust_Last_Name, COALESCE(Cust_Street_Address1,Cust_Street_Address2) AS Address
FROM Demo_Customers;
```

CUSTOMER_ID	CUST_LAST_NAME	ADDRESS
1	Dulles	45020 Aviation Drive
2	Hartsfield	6000 North Terminal Parkway
3	Logan	1 Harborside Drive
4	OHare	10000 West OHare
5	LaGuardia	Hangar Center
6	Lambert	10701 Lambert International Blvd.
7	Bradley	Schoephoester Road

this is especially useful to manage input error from the end-users.

In SQL Server the function is a little bit more interesting because you can choose what to return if all argument are null:

```
SELECT Customer_Id, Cust_Last_Name,
COALESCE(Cust_Street_Address1,Cust_Street_Address2,'Unkown') AS Address
FROM Demo_Customers;
```

7.1.4 SQL Elementary Maths functions

We will suppose here that the reader alread know the basic addition (+), substraction (-), multiplication (*), division (/) and power syntax (POWER(number,exponent).

7.1.4.1 SQL ROUND function

The ROUND() function is used to round a numeric field to the number of decimals specified.

SQL ROUND() Syntax:

```
SELECT ROUND(column_name,decimals) FROM table_name;
```

Here is a typical simple example:

```
SELECT Order_Id, Product_Name, Unit_Price, Quantity,
TO_CHAR(Unit_Price*Quantity*0.076,'fm9999999.90') AS NonRoundedTotal,
TO_CHAR(ROUND(Unit_Price*Quantity*0.076/0.05,0)*0.05,'fm9999999.90') AS RoundedTotal
FROM Demo_Product_Info
LEFT JOIN Demo_Order_Items
ON Demo_Product_Info.Product_Id=Demo_Order_Items.Product_Id;
```

ORDER_ID	PRODUCT_NAME	UNIT_PRICE	QUANTITY	NONROUNDEDTOTAL	ROUNDEDTOTAL
1	Business Shirt	50	10	38.00	38.00
1	Trousers	80	8	48.64	48.65
1	Jacket	150	5	57.00	57.00
2	Business Shirt	50	3	11.40	11.40
2	Trousers	80	3	18.24	18.25
2	Jacket	150	3	34.20	34.20
2	Blouse	60	3	13.68	13.70
2	Skirt	80	3	18.24	18.25
2	Ladies Shoes	120	2	18.24	18.25

7.1.4.2 SQL LOG function

For a real example of LOG( ) application with SQL and in finance see page 272.

7.1.5 SQL Elementary Statistical functions

Oracle is the most powerful RDMS for descriptive, inferential and hypothesis statistical tests. This is why all examples in this chapter will be done only with Oracle.

The descriptive statistics functions can for sure be mixed with GROUP BY, WHERE, JOINS, ... SQL statements and especially subqueries!

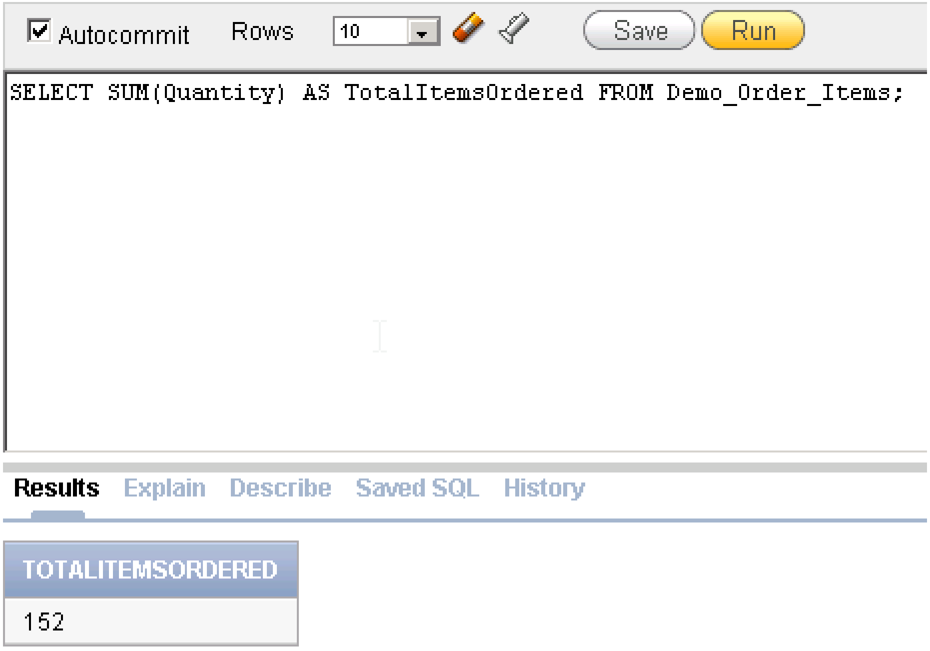
7.1.5.1 SQL SUM Function

The SUM() function returns the total sum of a numeric column.

SQL SUM() Syntax:

```
SELECT SUM(column_name) FROM table_name;
```

The following SQL statement finds the sum of all the Quantity fields for the Order Items table:



Or a little but more interesting:

```
SELECT Product_Id, SUM(Quantity) AS TotalItemsOrdered FROM Demo_Order_Items
GROUP BY Product_Id;
```

PRODUCT_ID	TOTALITEMSORDERED
1	23
6	12
2	21
4	16
5	14
8	16
3	18
7	11
9	10
10	11

etc...

Or an interesting one to get the Total Number of Records in ALL TABLES of a schema (**see page 273 for other metadata queries**):

```
select SUM(NUM_ROW) FROM USER_TABLES;
```

SUM(NUM_ROWS)
156

7.1.5.1.1 Running Total

Running totals are very typical request from many accouters and business analysts. Let us see how to do this using the SUM( ) function:

Autocommit Rows 10 Save Run

```
SELECT Ename, Sal,
SUM(Sal) OVER (ORDER BY Sal) as Running_Total
FROM EMP
ORDER BY Sal;
```

Results Explain Describe Saved SQL History

ENAME	SAL	RUNNING_TOTAL
SMITH	800	800
ADAMS	1100	1900
JAMES	1144.56	3044.56
MILLER	1300	4344.56
MARTIN	1506.01	7356.58
WARD	1506.01	7356.58
TURNER	1807.21	9163.79
ALLEN	1927.69	11091.48
CLARK	2450	13541.48
JONES	2975	16516.48

Nice to have! That was boring to write with previous versions!

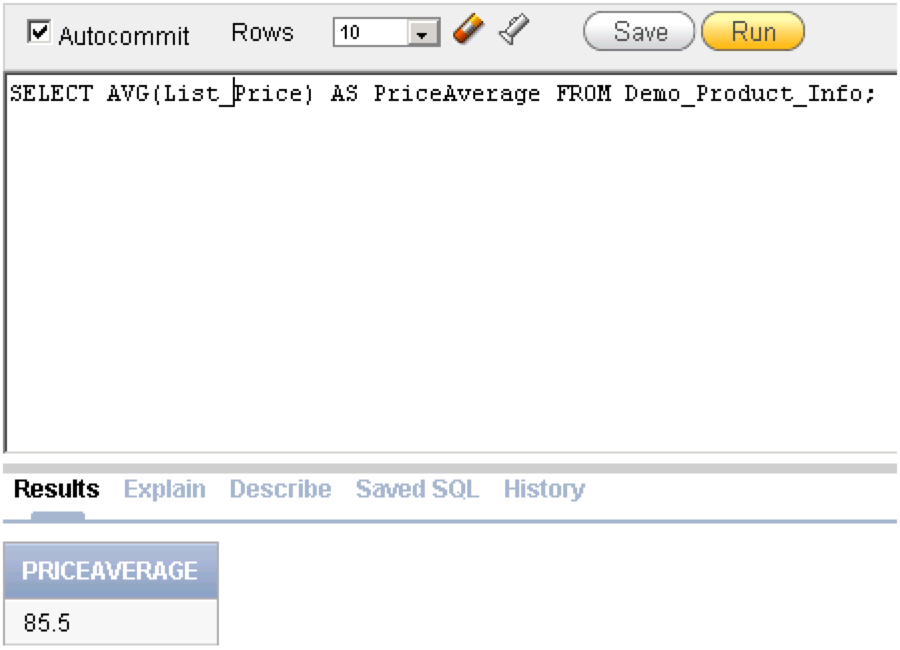
7.1.5.2 SQL Average Function

The AVG() function returns the average value of a numeric column.

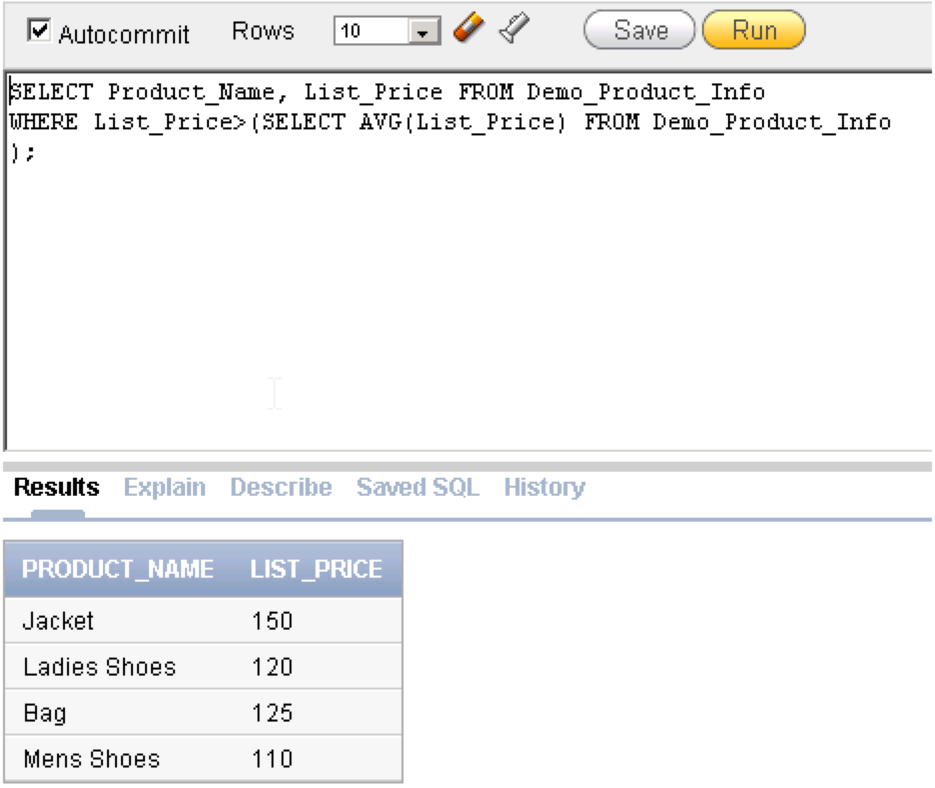
The syntax is the following:

```
SELECT AVG(column_name) FROM table_name
```

The following SQL statement gets the average value of the price column from the products table:



The following SQL statement selects the Product Name and Price records that have an above average price:


Autocommit
Rows
10
Save
Run
SELECT Product_Name, List_Price FROM Demo_Product_Info
WHERE List_Price>(SELECT AVG(List_Price) FROM Demo_Product_Info
);
Results
Explain
Describe
Saved SQL
History
PRODUCT_NAME
LIST_PRICE
Jacket
150
Ladies Shoes
120
Bag
125
Mens Shoes
110

7.1.5.3 SQL COUNT Function

The COUNT() function returns the average value of a numeric column.

The syntax is the following:

```
SELECT COUNT(column_name) FROM table_name
SELECT COUNT(column_name) FROM table_name;
```

The COUNT(*) function returns the number of records in a table:

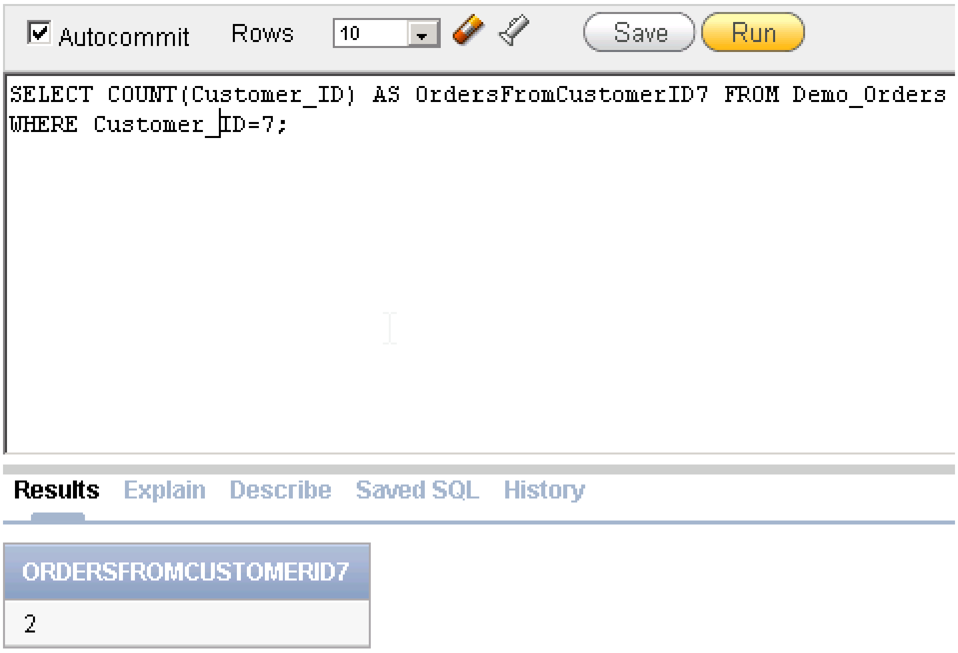
```
SELECT COUNT(*) FROM table_name;
```

The COUNT(DISTINCT column_name) function returns the number of distinct values of the specified column:

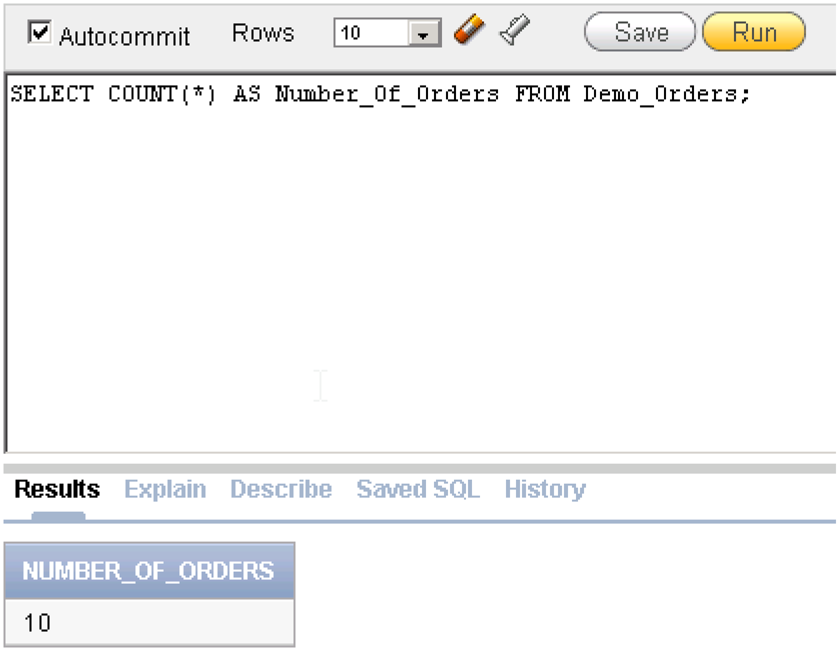
```
SELECT COUNT(DISTINCT column_name) FROM table_name;
```

Note: COUNT(DISTINCT) works with ORACLE and Microsoft SQL Server, but not with Microsoft Access.

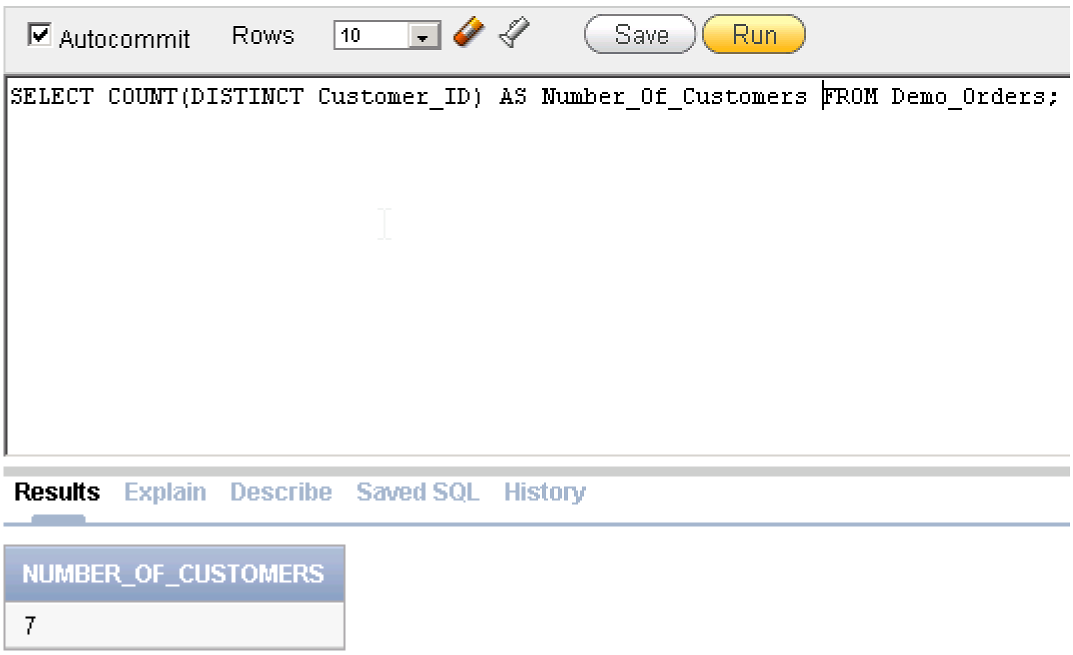
The following SQL statement counts the number of orders from CustomerID 7 from the Orders table:



The following SQL statement counts the total number of orders in the Orders table:



The following SQL statement counts the number of unique customers in the Orders table:



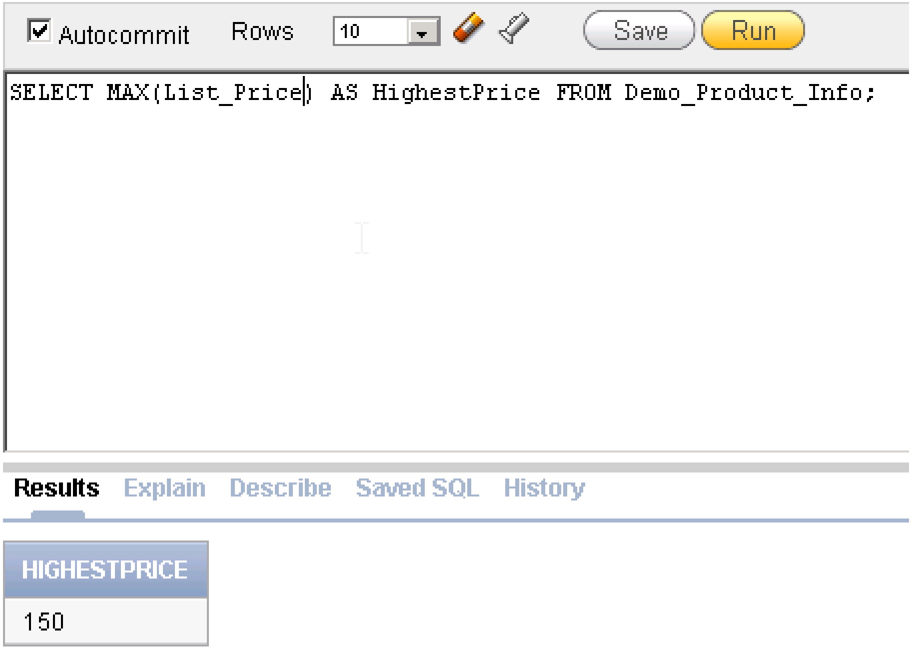
7.1.5.4 SQL MAX/MIN function

The MAX()/MIN() function returns the largest value of the selected column.

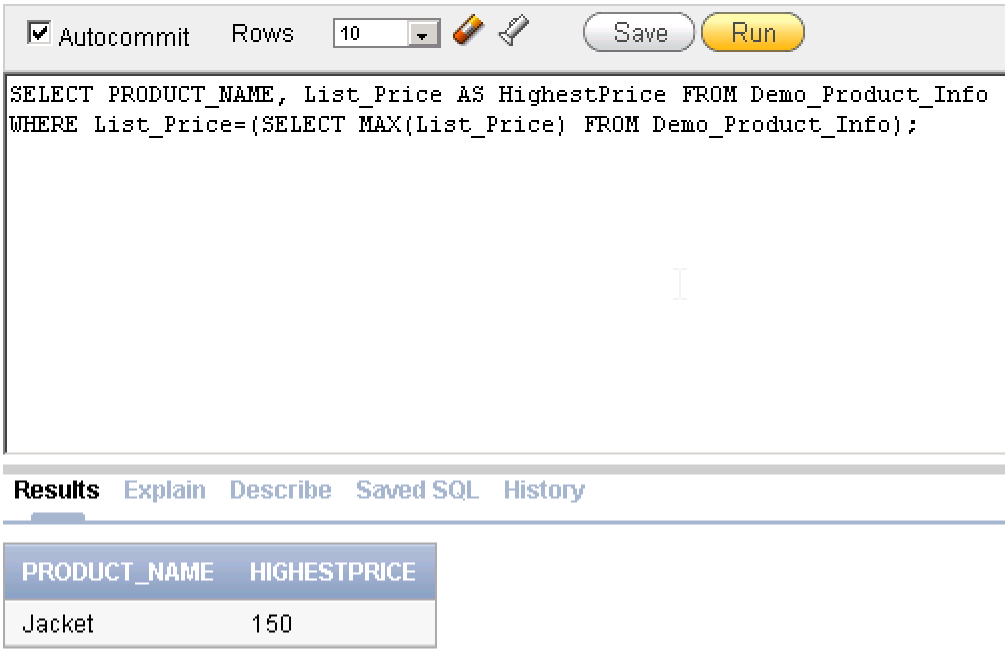
SQL MAX()/MIN() Syntax:

```
SELECT MAX(column_name) FROM table_name;
```

The following SQL statement gets the largest value of the Price column from the Products table:



Or with the corresponding informations:



and just replace MAX( ) by MIN( ) in the above queries to see how MIN( ) works.

7.1.5.5 SQL MEDIAN Function

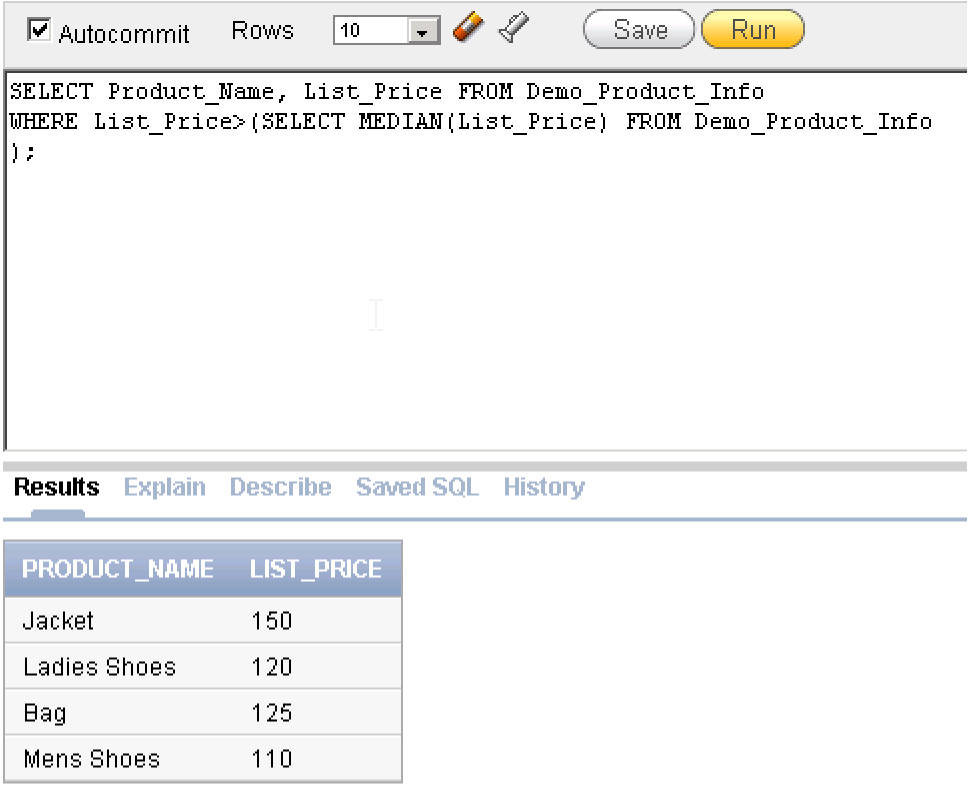
The MEDIAN( ) function returns the Median value of a numeric column.

The syntax is the following:

```
SELECT MEDIAN(column_name) FROM table_name
```

The following SQL statement gets the average value of the price column from the products table:

The following SQL statement selects the Product Name and Price records that have an above median price:


Autocommit
Rows
10
Save
Run
SELECT Product_Name, List_Price FROM Demo_Product_Info
WHERE List_Price>(SELECT MEDIAN(List_Price) FROM Demo_Product_Info
);
Results
Explain
Describe
Saved SQL
History
PRODUCT_NAME
LIST_PRICE
Jacket
150
Ladies Shoes
120
Bag
125
Mens Shoes
110

7.1.5.6 SQL Continuous Percentiles

PERCENTILE_CONT( ) is an inverse distribution function that assumes a continuous distribution model. It takes a percentile value and a sort specification, and returns an interpolated value that would fall into that percentile value with respect to the sort specification.

Here is a first example:

```
SELECT ename, Sal, DeptNo,
PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY Sal DESC)
OVER (PARTITION BY 1) AS Median,
PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY Sal DESC)
OVER (PARTITION BY 1) AS First_Quartile,
PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY Sal DESC)
OVER (PARTITION BY 1) AS Third_Quartile
FROM Emp;
```

ENAME	SAL	DEPTNO	MEDIAN	FIRST_QUARTILE	THIRD_QUARTILE
SMITH	800	20	1867.45	1351.5025	2993.75
ADAMS	1100	20	1867.45	1351.5025	2993.75
JAMES	1144.56	30	1867.45	1351.5025	2993.75
MILLER	1300	10	1867.45	1351.5025	2993.75
MARTIN	1506.01	30	1867.45	1351.5025	2993.75
WARD	1506.01	30	1867.45	1351.5025	2993.75
TURNER	1807.21	30	1867.45	1351.5025	2993.75
ALLEN	1927.69	30	1867.45	1351.5025	2993.75
CLARK	2450	10	1867.45	1351.5025	2993.75
JONES	2975	20	1867.45	1351.5025	2993.75
FORD	3000	20	1867.45	1351.5025	2993.75
SCOTT	3000	20	1867.45	1351.5025	2993.75
BLAKE	3433.69	30	1867.45	1351.5025	2993.75
KING	5000	10	1867.45	1351.5025	2993.75

Or the same statistics by department:

Autocommit Rows 20 Save Run

```sql
SELECT ename, Sal, DeptNo,
PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY Sal DESC)
OVER (PARTITION BY DeptNo) AS Median,
PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY Sal DESC)
OVER (PARTITION BY DeptNo) AS First_Quartile,
PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY Sal DESC)
OVER (PARTITION BY DeptNo) AS Third_Quartile
FROM Emp;
```

Results Explain Describe Saved SQL History

ENAME	SAL	DEPTNO	MEDIAN	FIRST_QUARTILE	THIRD_QUARTILE
MILLER	1300	10	2450	1875	3725
CLARK	2450	10	2450	1875	3725
KING	5000	10	2450	1875	3725
SMITH	800	20	2975	1100	3000
ADAMS	1100	20	2975	1100	3000
JONES	2975	20	2975	1100	3000
SCOTT	3000	20	2975	1100	3000
FORD	3000	20	2975	1100	3000
JAMES	1144.56	30	1656.61	1506.01	1897.57
MARTIN	1506.01	30	1656.61	1506.01	1897.57
WARD	1506.01	30	1656.61	1506.01	1897.57
TURNER	1807.21	30	1656.61	1506.01	1897.57
ALLEN	1927.69	30	1656.61	1506.01	1897.57
BLAKE	3433.69	30	1656.61	1506.01	1897.57

7.1.5.7 SQL Discrete Percentiles

PERCENTILE_DISC( ) is an inverse distribution function that assumes a discrete distribution model. It takes a percentile value and a sort specification and returns an element from the set (**there is no interpolation!**: then it takes the nearest value of the set).

It is interesting to take for example the previous one but now with the discrete percentile:

Autocommit Rows 20 Save Run

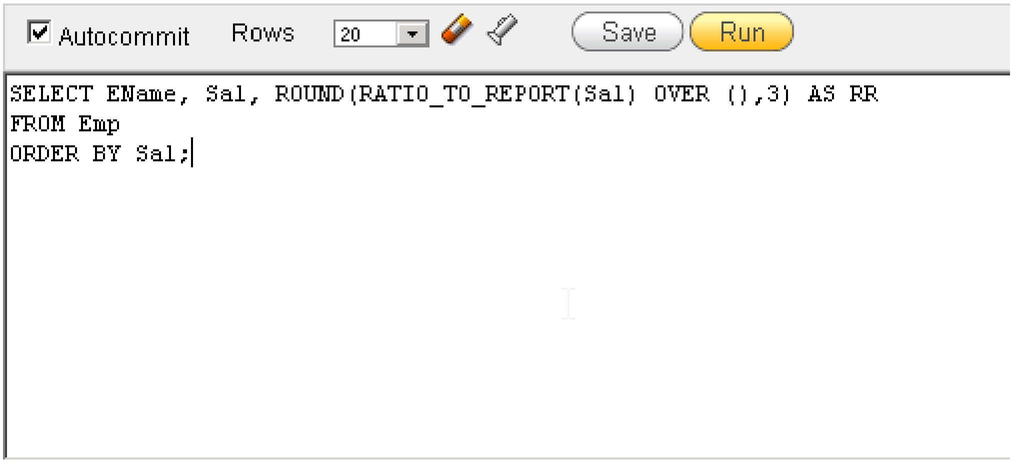
```sql
SELECT ename, Sal, DeptNo,
PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY Sal DESC)
OVER (PARTITION BY DeptNo) AS Median,
PERCENTILE_DISC(0.75) WITHIN GROUP (ORDER BY Sal DESC)
OVER (PARTITION BY DeptNo) AS First_Quartile,
PERCENTILE_DISC(0.25) WITHIN GROUP (ORDER BY Sal DESC)
OVER (PARTITION BY DeptNo) AS Third_Quartile
FROM Emp;
```

Results Explain Describe Saved SQL History

ENAME	SAL	DEPTNO	MEDIAN	FIRST_QUARTILE	THIRD_QUARTILE
MILLER	1300	10	2450	1300	5000
CLARK	2450	10	2450	1300	5000
KING	5000	10	2450	1300	5000
SMITH	800	20	2975	1100	3000
ADAMS	1100	20	2975	1100	3000
JONES	2975	20	2975	1100	3000
SCOTT	3000	20	2975	1100	3000
FORD	3000	20	2975	1100	3000
JAMES	1144.56	30	1807.21	1506.01	1927.69
MARTIN	1506.01	30	1807.21	1506.01	1927.69
WARD	1506.01	30	1807.21	1506.01	1927.69
TURNER	1807.21	30	1807.21	1506.01	1927.69
ALLEN	1927.69	30	1807.21	1506.01	1927.69
BLAKE	3433.69	30	1807.21	1506.01	1927.69

7.1.5.8 SQL Ratio to Report

The RATIO_TO_REPORT( ) computes the ratio of a value to the sum of a set of values:

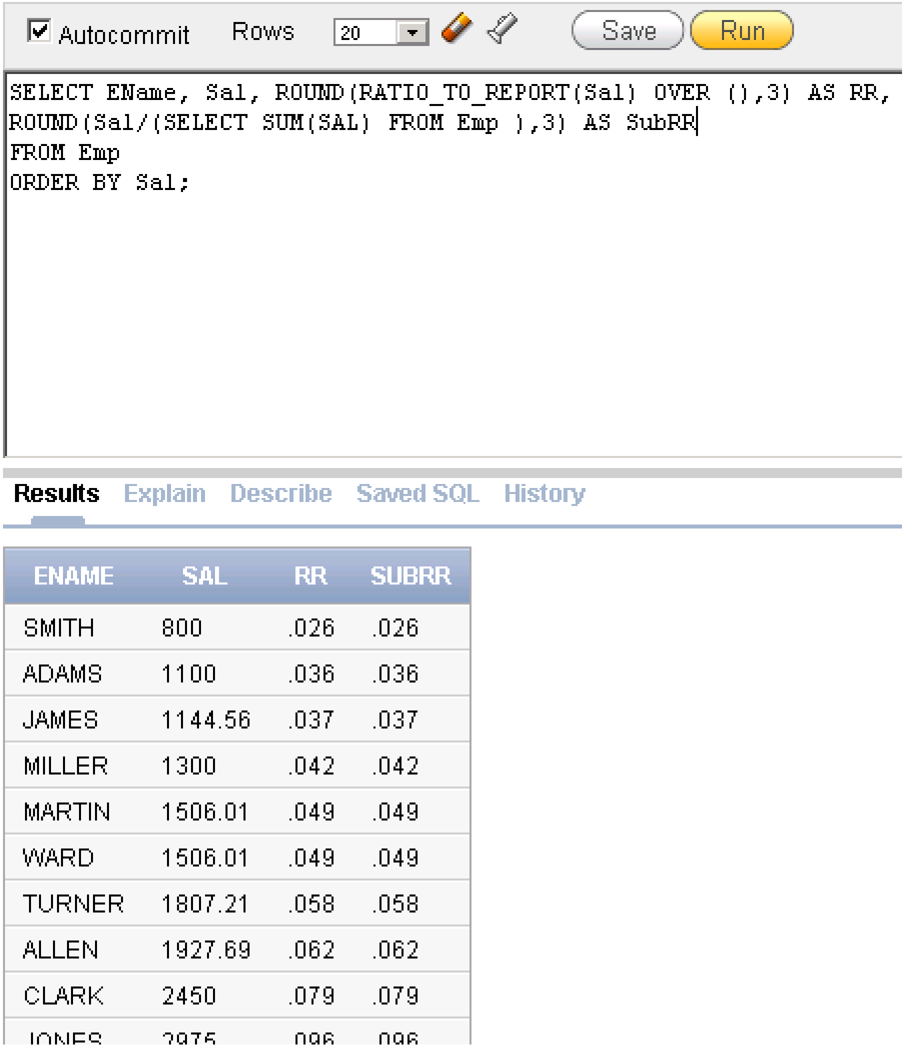



```sql
SELECT EName, Sal, ROUND(RATIO_TO_REPORT(Sal) OVER (),3) AS RR
FROM Emp
ORDER BY Sal;
```

Results Explain Describe Saved SQL History

ENAME	SAL	RR
SMITH	800	.026
ADAMS	1100	.036
JAMES	1144.56	.037
MILLER	1300	.042
MARTIN	1506.01	.049
WARD	1506.01	.049
TURNER	1807.21	.058
ALLEN	1927.69	.062
CLARK	2450	.079
JONES	2975	.096
FORD	3000	.097
SCOTT	3000	.097
BLAKE	3433.69	.111
KING	5000	.162

This is the same as a simple subquery likes w show in the next screenshot:



ENAME	SAL	RR	SUBRR
SMITH	800	.026	.026
ADAMS	1100	.036	.036
JAMES	1144.56	.037	.037
MILLER	1300	.042	.042
MARTIN	1506.01	.049	.049
WARD	1506.01	.049	.049
TURNER	1807.21	.058	.058
ALLEN	1927.69	.062	.062
CLARK	2450	.079	.079
JONES	2975	.096	.096

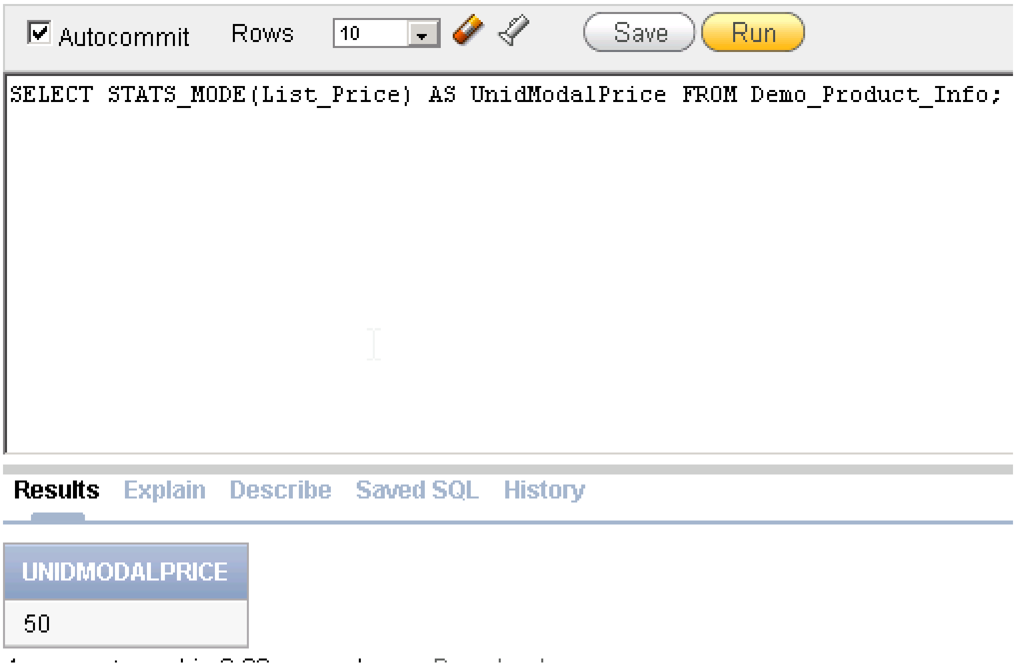
7.1.5.9 SQL Mode (unimodal) Function

The STATS_MODE( ) function returns the Mode value of a numeric column.

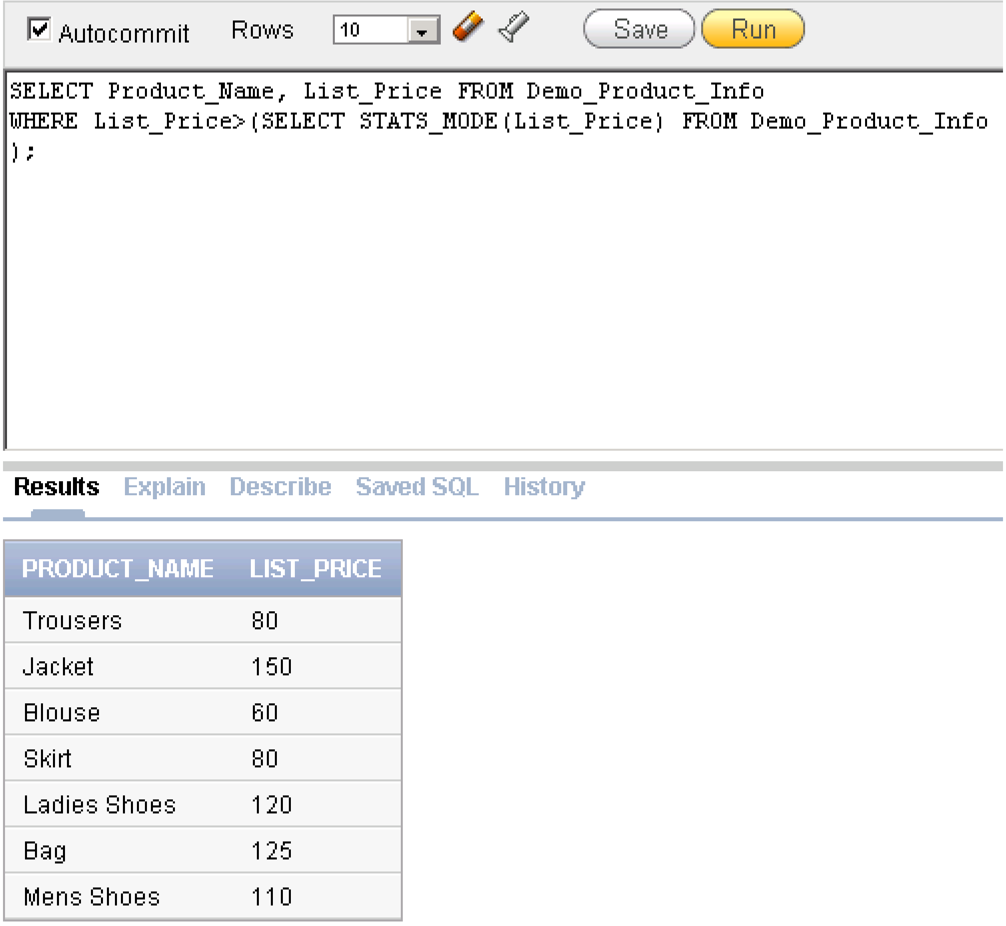
The syntax is the following:

```
SELECT STATS_MODE(column_name) FROM table_name
```

The following SQL statement gets the average value of the price column from the products table:



The following SQL statement selects the Product Name and Price records that have an above modal price:




```
SELECT Product_Name, List_Price FROM Demo_Product_Info
WHERE List_Price>(SELECT STATS_MODE(List_Price) FROM Demo_Product_Info
);
```

PRODUCT_NAME	LIST_PRICE
Trousers	80
Jacket	150
Blouse	60
Skirt	80
Ladies Shoes	120
Bag	125
Mens Shoes	110

7.1.5.10 SQL pooled Standard Deviation and Variance

A funny example of the use of standard deviation STDDEV( ) and variance VARIANCE( ) by grouping sales based on ISO week numbering:

Autocommit Rows 10 Save Run

```
SELECT TO_CHAR(Order_TimeStamp,'IW') AS ISO_Week_Nb,
SUM(Unit_Price*Quantity) AS Sum, ROUND(AVG(Unit_Price*Quantity),2) AS Average,
MEDIAN(Unit_Price*Quantity) AS Median,
ROUND(STDDEV(Unit_Price*Quantity),2) AS Standard_Deviation,
ROUND(VARIANCE(Unit_Price*Quantity),2) AS Variance
FROM Demo_Orders
INNER JOIN Demo_Order_Items
ON Demo_Orders.Order_Id=Demo_Order_Items.Order_Id
GROUP BY TO_CHAR(Order_TimeStamp,'IW');
```

Results Explain Describe Saved SQL History

ISO_WEEK_NB	SUM	AVERAGE	MEDIAN	STANDARD_DEVIATION	VARIANCE
30	870	290	300	36.06	1300
32	730	243.33	240	5.77	33.33
33	1060	265	245	153.3	23500
34	905	129.29	125	28.35	803.57
35	1515	378.75	367.5	51.05	2606.25
36	2040	204	190	48.12	2315.56
38	4020	268	240	150.06	22517.14
39	1890	630	640	125.3	15700

7.1.5.10.1 Population Standard Deviation and Variance

Oracle also has the functions:

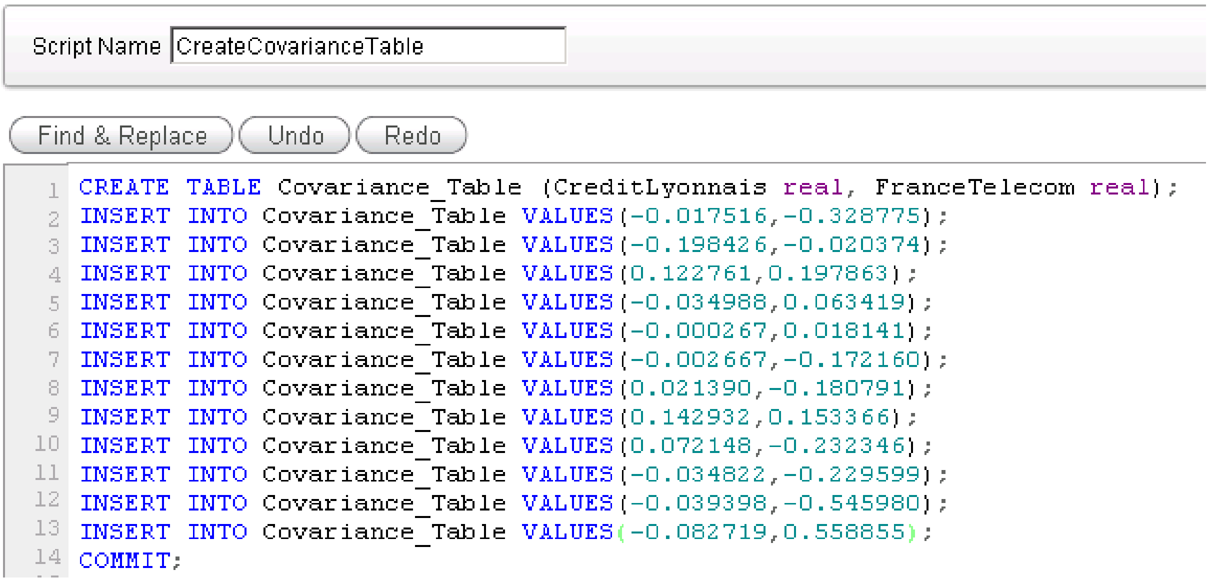
STDEV_POP and VAR_POP

7.1.5.11 SQL Sample Covariance

Returns the sample covariance of a set of number pairs. Oracle can't without PL/SQL return the variance-covariance matrix.

Note: In statistics we know (see Statistics course) that de regression matrix is more interesting but the variance-covariance matrix is still useful in financial modeling.

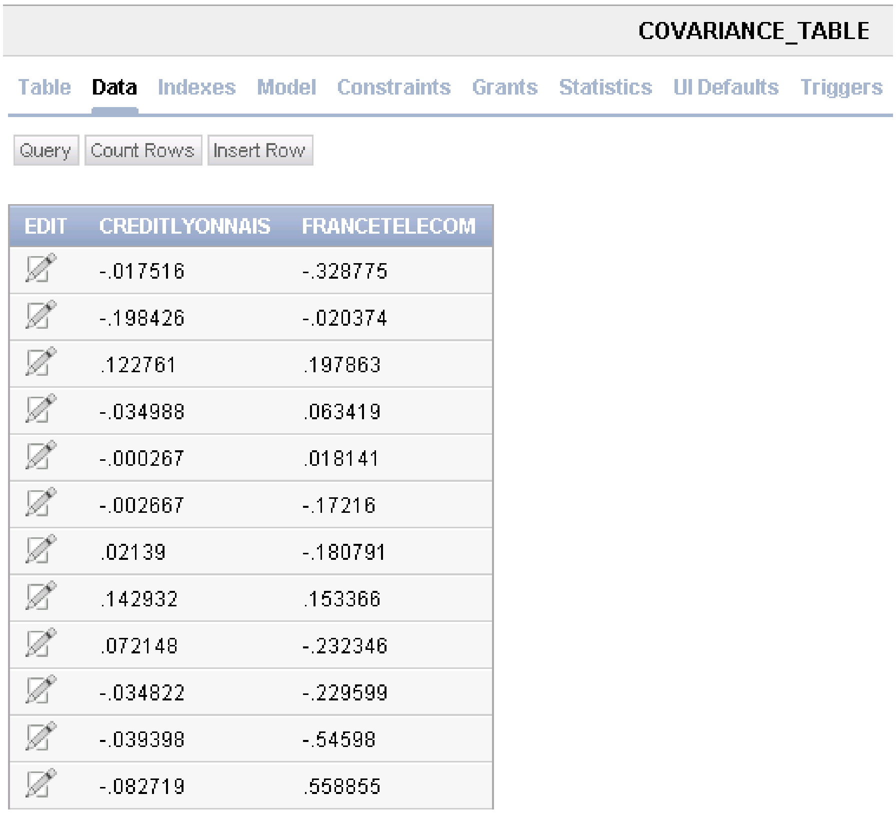
Then here to see an example we will first create a two columns table with the following script:



and here is the corresponding text (for copy/paste purpose during the training):

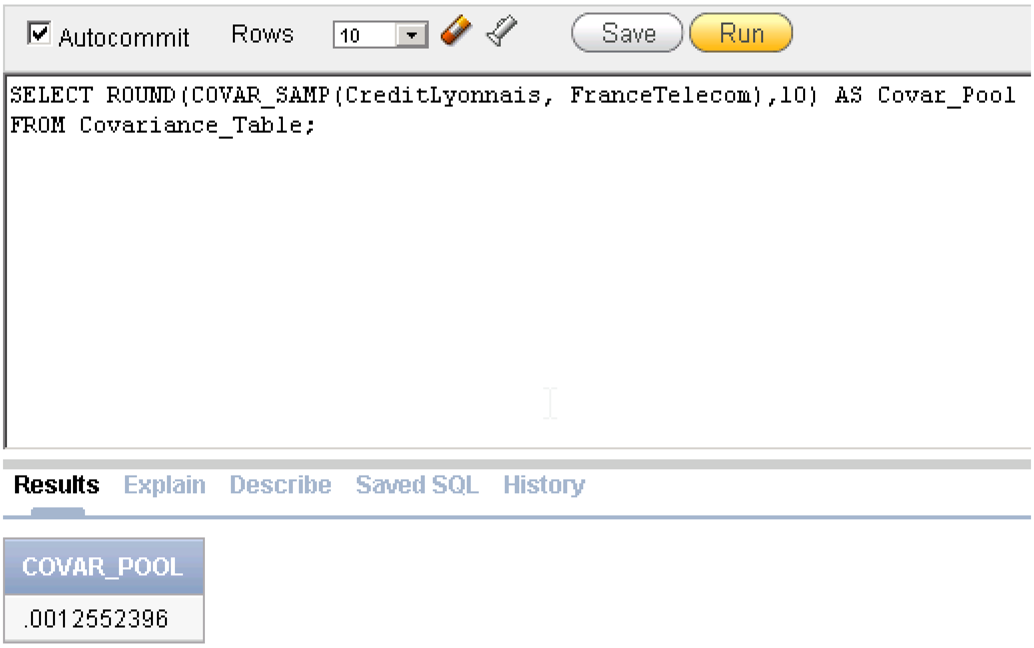
```
CREATE TABLE Covariance_Table (CreditLyonnais real, FranceTelecom real);
INSERT INTO Covariance_Table VALUES(-0.017516,-0.328775);
INSERT INTO Covariance_Table VALUES(-0.198426,-0.020374);
INSERT INTO Covariance_Table VALUES(0.122761,0.197863);
INSERT INTO Covariance_Table VALUES(-0.034988,0.063419);
INSERT INTO Covariance_Table VALUES(-0.000267,0.018141);
INSERT INTO Covariance_Table VALUES(-0.002667,-0.172160);
INSERT INTO Covariance_Table VALUES(0.021390,-0.180791);
INSERT INTO Covariance_Table VALUES(0.142932,0.153366);
INSERT INTO Covariance_Table VALUES(0.072148,-0.232346);
INSERT INTO Covariance_Table VALUES(-0.034822,-0.229599);
INSERT INTO Covariance_Table VALUES(-0.039398,-0.545980);
INSERT INTO Covariance_Table VALUES(-0.082719,0.558855);
COMMIT;
```

This gives us a part of the table used in Minitab, Tanagra, SPSS and R training:



EDIT	CREDITLYONNAIS	FRANCETELECOM
	-.017516	-.328775
	-.198426	-.020374
	.122761	.197863
	-.034988	.063419
	-.000267	.018141
	-.002667	-.17216
	.02139	-.180791
	.142932	.153366
	.072148	-.232346
	-.034822	-.229599
	-.039398	-.54598
	-.082719	.558855

and now we run our query:




```sql
SELECT ROUND(COVAR_SAMP(CreditLyonnais, FranceTelecom),10) AS Covar_Pool
FROM Covariance_Table;
```

COVAR_POOL
.0012552396

To compare with the value obtained with Minitab:

Covariances : Crédit Lyonn; France Téléc; Lafarge; Saint-Gobain; Total Fina

	Crédit Lyonnais	France Télécom	Lafarge
Crédit Lyonnais	0.00831448		
France Télécom	0.00125522	0.08290560	
Lafarge	0.00443929	-0.00073856	0.00440680
Saint-Gobain	0.00536049	-0.01575319	0.00537174
Total Fina	0.00371970	-0.00312211	0.00241502

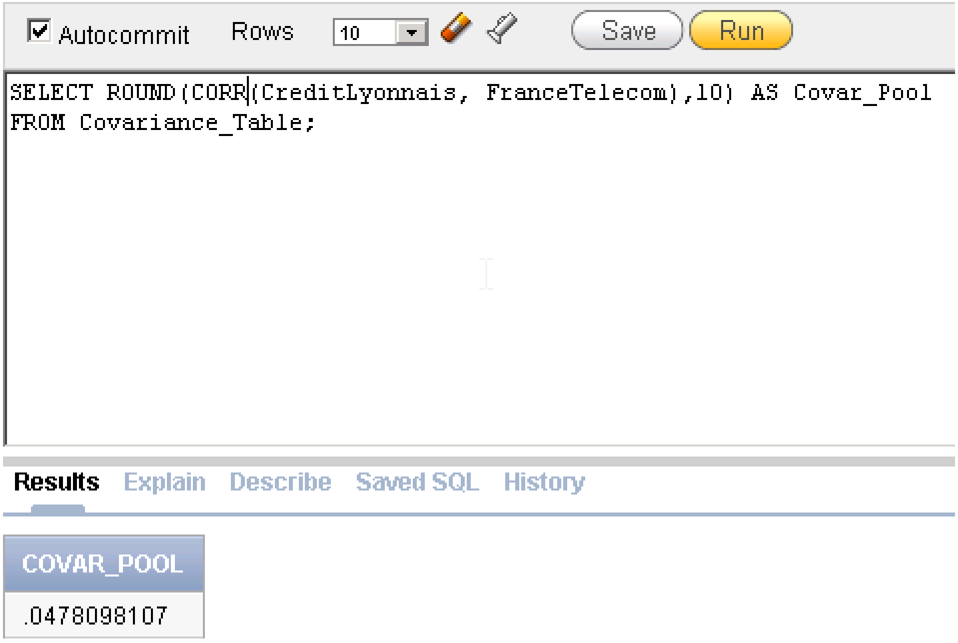
	Saint-Gobain	Total Fina
Saint-Gobain	0.01198378	
Total Fina	0.00445611	0.00473239

Everything is fine!

7.1.5.12 SQL Pearson Correlation

The Person correlation is for sure used a lot in Finance but also anywhere else where linear models are studied.

The table we will use here is the same as the one for the Sample Covariance (see above). Then we just run the following query:



and we compare with Minitab:

Affichage des données

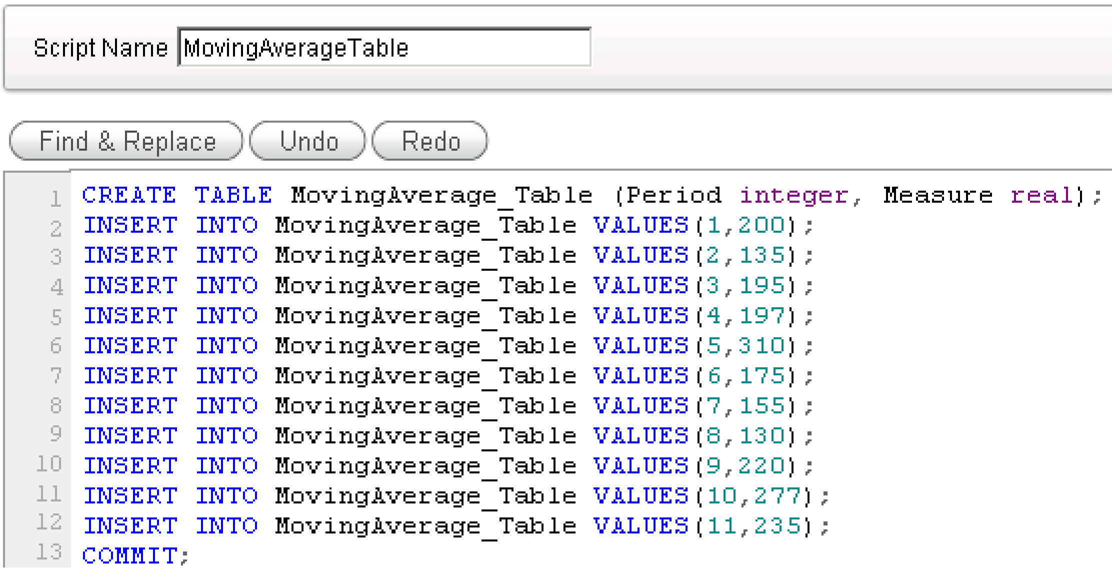
Matrice CORRELATI01

1.00000	0.04781	0.73339	0.53702	0.59299
0.04781	1.00000	-0.03864	-0.49978	-0.15762
0.73339	-0.03864	1.00000	0.73919	0.52883
0.53702	-0.49978	0.73919	1.00000	0.59172
0.59299	-0.15762	0.52883	0.59172	1.00000

Everything is fine!

7.1.5.13 SQL Moving Average

To see how works moving average with Oracle we will first create a table with the following script:

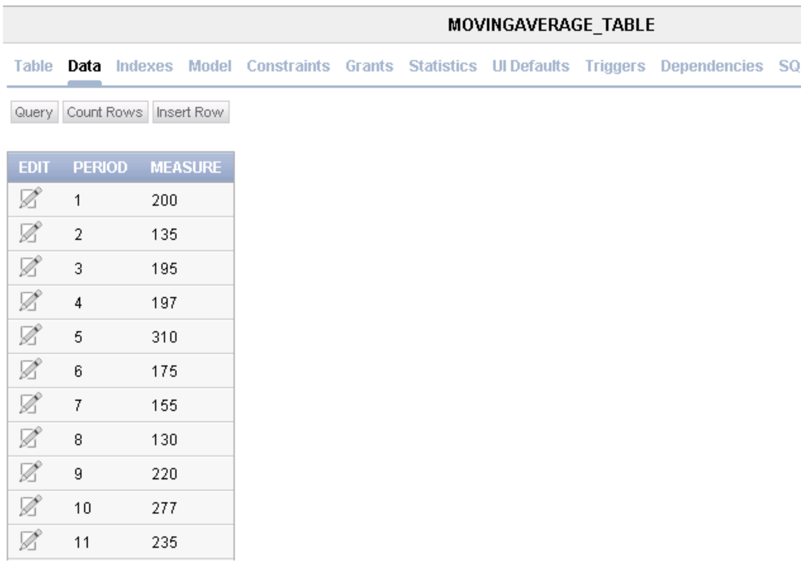



```sql
CREATE TABLE MovingAverage_Table (Period integer, Measure real);
INSERT INTO MovingAverage_Table VALUES(1,200);
INSERT INTO MovingAverage_Table VALUES(2,135);
INSERT INTO MovingAverage_Table VALUES(3,195);
INSERT INTO MovingAverage_Table VALUES(4,197);
INSERT INTO MovingAverage_Table VALUES(5,310);
INSERT INTO MovingAverage_Table VALUES(6,175);
INSERT INTO MovingAverage_Table VALUES(7,155);
INSERT INTO MovingAverage_Table VALUES(8,130);
INSERT INTO MovingAverage_Table VALUES(9,220);
INSERT INTO MovingAverage_Table VALUES(10,277);
INSERT INTO MovingAverage_Table VALUES(11,235);
COMMIT;
```

and here is the corresponding text (for copy/paste purpose during the training):

CREATE TABLE MovingAverage_Table (Period integer, Measure real);
INSERT INTO MovingAverage_Table VALUES(1,200);
INSERT INTO MovingAverage_Table VALUES(2,135);
INSERT INTO MovingAverage_Table VALUES(3,195);
INSERT INTO MovingAverage_Table VALUES(4,197);
INSERT INTO MovingAverage_Table VALUES(5,310);
INSERT INTO MovingAverage_Table VALUES(6,175);
INSERT INTO MovingAverage_Table VALUES(7,155);
INSERT INTO MovingAverage_Table VALUES(8,130);
INSERT INTO MovingAverage_Table VALUES(9,220);
INSERT INTO MovingAverage_Table VALUES(10,277);
INSERT INTO MovingAverage_Table VALUES(11,235);
COMMIT;

This gives us the table used in Minitab, SPSS and R training:



EDIT	PERIOD	MEASURE
	1	200
	2	135
	3	195
	4	197
	5	310
	6	175
	7	155
	8	130
	9	220
	10	277
	11	235

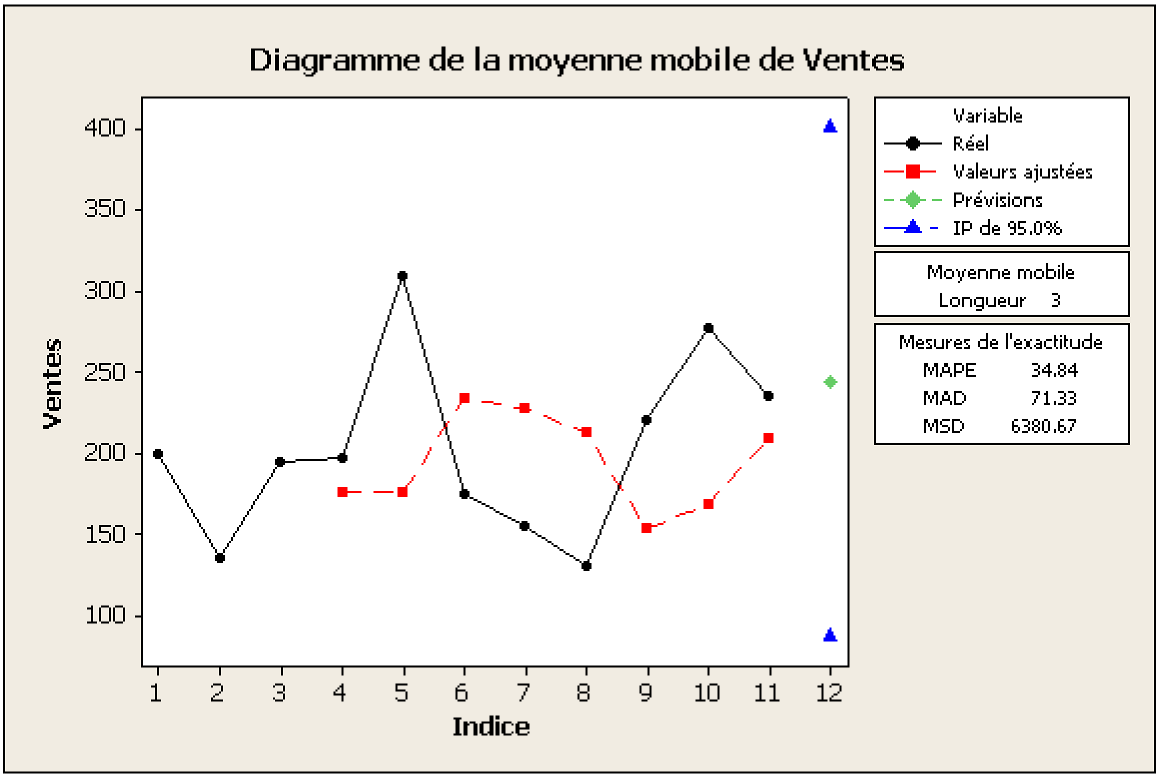
and now we run the following query:

```
SELECT Period, Measure, ROUND(AVG(Measure)
OVER (ORDER BY Period ROWS BETWEEN 2 PRECEDING AND CURRENT ROW),2)
AS Moving_Average FROM MovingAverage_Table;
```

PERIOD	MEASURE	MOVING_AVERAGE
1	200	200
2	135	167.5
3	195	176.67
4	197	175.67
5	310	234
6	175	227.33
7	155	213.33
8	130	153.33
9	220	168.33
10	277	209
11	235	244

and we compare with the 3 MA analysis in Minitab:

↓	C1	C2	C3
	Ventes	MOY MOB1	PREVISIONS1
1	200	*	244
2	135	*	
3	195	176.667	
4	197	175.667	
5	310	234.000	
6	175	227.333	
7	155	213.333	
8	130	153.333	
9	220	168.333	
10	277	209.000	
11	235	244.000	

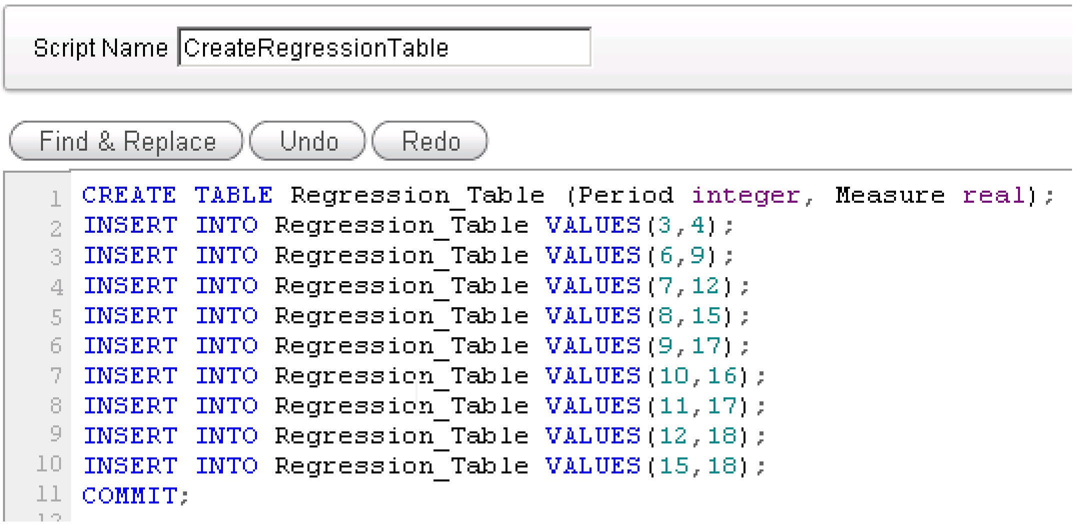


excepted the chart, everything is fine :-)

7.1.5.14 SQL Linear Regression

The linear regression functions fit an ordinary-least-squares regression line to a set of number pairs. You can use them as both aggregate and analytic functions.

To see how works regression functions with Oracle we will first create a table with the following script:

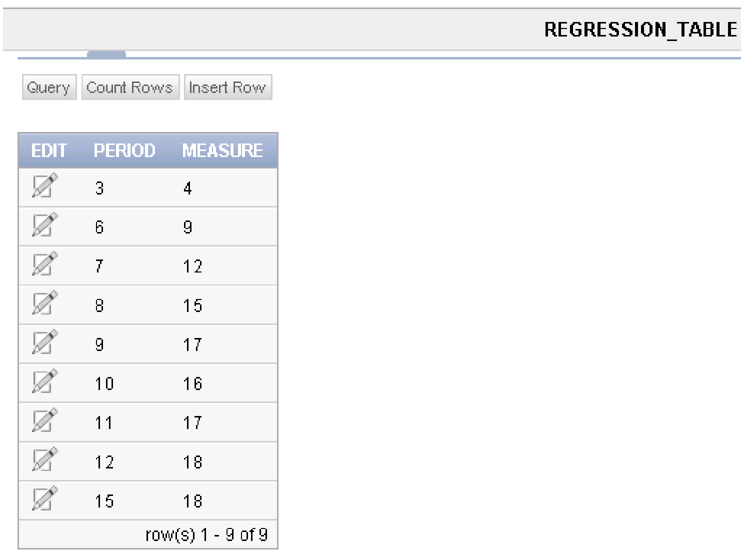



```
CREATE TABLE Regression_Table (Period integer, Measure real);
INSERT INTO Regression_Table VALUES(3,4);
INSERT INTO Regression_Table VALUES(6,9);
INSERT INTO Regression_Table VALUES(7,12);
INSERT INTO Regression_Table VALUES(8,15);
INSERT INTO Regression_Table VALUES(9,17);
INSERT INTO Regression_Table VALUES(10,16);
INSERT INTO Regression_Table VALUES(11,17);
INSERT INTO Regression_Table VALUES(12,18);
INSERT INTO Regression_Table VALUES(15,18);
COMMIT;
```

and here is the corresponding text (for copy/paste purpose during the training):

```
CREATE TABLE Regression_Table (Period integer, Measure real);
INSERT INTO Regression_Table VALUES(3,4);
INSERT INTO Regression_Table VALUES(6,9);
INSERT INTO Regression_Table VALUES(7,12);
INSERT INTO Regression_Table VALUES(8,15);
INSERT INTO Regression_Table VALUES(9,17);
INSERT INTO Regression_Table VALUES(10,16);
INSERT INTO Regression_Table VALUES(11,17);
INSERT INTO Regression_Table VALUES(12,18);
INSERT INTO Regression_Table VALUES(15,18);
COMMIT;
```

This gives us the table used in Minitab, Tanagra, SPSS and R training:



EDIT	PERIOD	MEASURE
	3	4
	6	9
	7	12
	8	15
	9	17
	10	16
	11	17
	12	18
	15	18

row(s) 1 - 9 of 9

Then we run the following query:

Autocommit Rows 10 Save Run

```
SELECT DISTINCT
ROUND(REGR_SLOPE(Measure,Period)
OVER (PARTITION BY 1),5) AS Slope,
ROUND(REGR_INTERCEPT(Measure,Period)
OVER (PARTITION BY 1),5) AS Intercept,
ROUND(REGR_R2(Measure,Period)
OVER (PARTITION BY 1),5) AS R2_Pearson,
REGR_COUNT(Measure,Period)
OVER (PARTITION BY 1) count,
REGR_AVGX(Measure,Period)
OVER (PARTITION BY 1) avgx,
REGR_AVGY(Measure,Period)
OVER (PARTITION BY 1) avgy
FROM Regression_Table;
```

Results Explain Describe Saved SQL History

SLOPE	INTERCEPT	R2_PEARSON	COUNT	AVGX	AVGY
1.22	3.02	.80891	9	9	14

and we compare with Minitab:

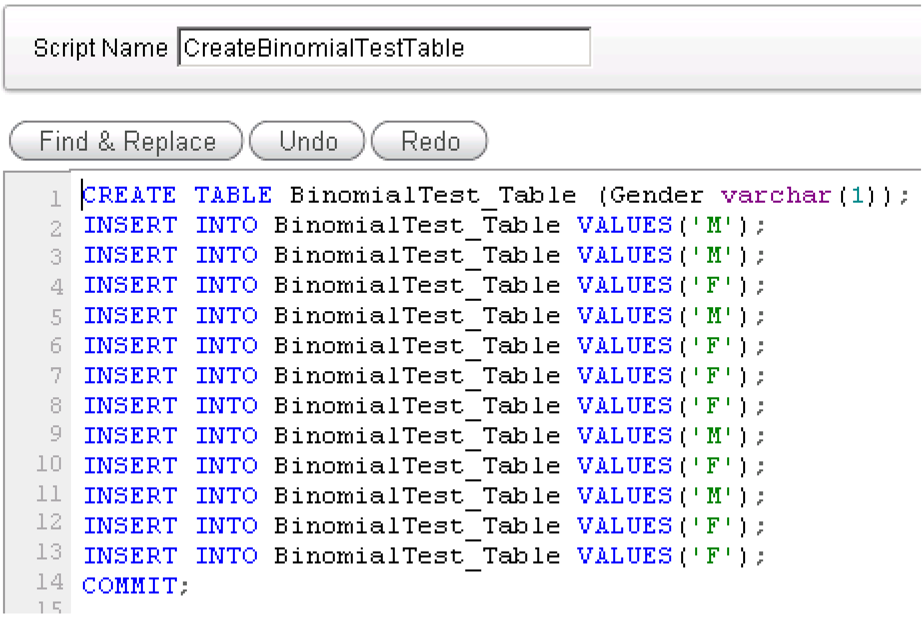
```
L'équation de régression est
Ventes = 3.02 + 1.22 Mois


Prédicteur    Coeff  Coef ErT     T      P
Constante     3.020     2.151  1.40  0.203
Mois         1.2200    0.2241  5.44  0.001


S = 2.24117   R carré = 80.9 %   R carré (ajust) = 78.2 %


Analyse de variance

                       Somme des
Source             DL     carrés      CM      F      P
Régression          1     148.84  148.84  29.63  0.001
Erreur résiduelle   7      35.16    5.02
Total               8     184.00


Observations aberrantes

                                                           Valeur
                             Valeur              Valeur  résiduelle
Observation  Mois  Ventes   ajustée  Ajust ErT  résiduelle  normalisée
          9  15.0  18.000    21.320      1.538      -3.320      -2.04R

R indique une observation ayant une valeur résiduelle normalisée importante
```

for sure a statistical software gives more results but otherwise what we get back we Oracle seems OK!

7.1.5.15 SQL Binomial test

STATS_BINOMIAL_TEST( ) is an exact probability test used for dichotomous variables, where only two possible values exist. It tests the difference between a sample proportion and a given proportion. The sample size in such tests is usually small.

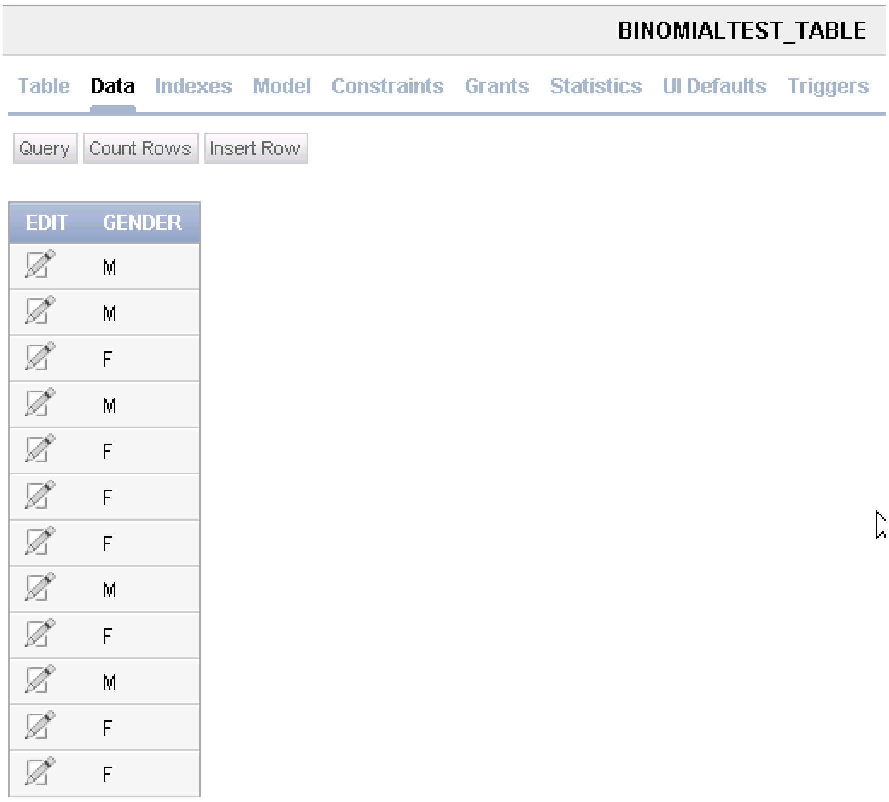
To see how works regression functions with Oracle we will first create a table with the following script:



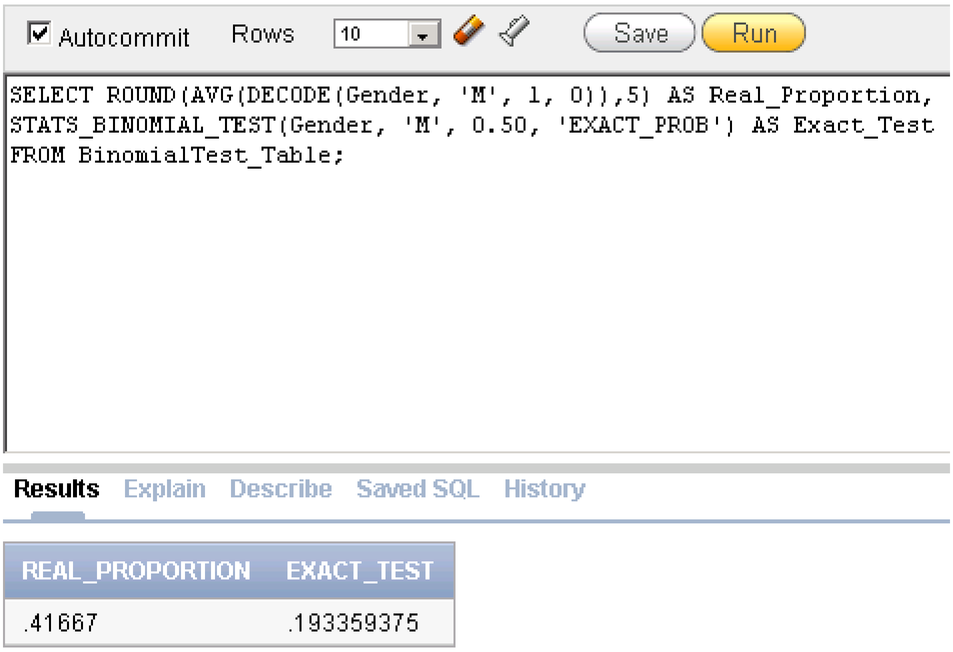
and here is the corresponding text (for copy/paste purpose during the training):

```
CREATE TABLE BinomialTest_Table (Gender varchar(1));
INSERT INTO BinomialTest_Table VALUES('M');
INSERT INTO BinomialTest_Table VALUES('M');
INSERT INTO BinomialTest_Table VALUES('F');
INSERT INTO BinomialTest_Table VALUES('M');
INSERT INTO BinomialTest_Table VALUES('F');
INSERT INTO BinomialTest_Table VALUES('F');
INSERT INTO BinomialTest_Table VALUES('F');
INSERT INTO BinomialTest_Table VALUES('M');
INSERT INTO BinomialTest_Table VALUES('F');
INSERT INTO BinomialTest_Table VALUES('M');
INSERT INTO BinomialTest_Table VALUES('F');
INSERT INTO BinomialTest_Table VALUES('F');
COMMIT;
```

This gives us the table used in Minitab, SPSS and R training:



Then we run first the following query:



This is correct. It gives the exact probability of having 5 Mens under the hypothesis that founding a Man or a Women is equal (=50%). This is corresponding with our calculation made with Microsoft Office Excel in the Statistical course:

Now we run the following query:

```
SELECT ROUND(AVG(DECODE(Gender, 'M', 1, 0)),5) AS Real_Proportion,
ROUND(STATS_BINOMIAL_TEST(Gender, 'M', 0.50, 'EXACT_PROB'),5) AS Exact_Test,
ROUND(STATS_BINOMIAL_TEST(Gender, 'M', 0.50, 'ONE_SIDED_PROB_OR_LESS'),5) AS One_Sided_Left_Test
FROM BinomialTest_Table;
```

REAL_PROPORTION	EXACT_TEST	ONE_SIDED_LEFT_TEST
.41667	.19336	.38721

This is correct. It gives the exact probability of having 5 Mens or less than under the hypothesis that founding a Man or a Women is equal (=50%). This is corresponding with our calculation made with Microsoft Office Excel in the Statistical course:

A1	=LOI.BINOMIALE(5;12;0.5;1)
	A
1	0.387207

Now we run the following query:

```
SELECT ROUND(AVG(DECODE(Gender, 'M', 1, 0)),5) AS Real_Proportion,
ROUND(STATS_BINOMIAL_TEST(Gender, 'M', 0.50, 'EXACT_PROB'),5) AS Exact_Test,
ROUND(STATS_BINOMIAL_TEST(Gender, 'M', 0.50, 'ONE_SIDED_PROB_OR_LESS'),5) AS One_Sided_Left_Test,
ROUND(STATS_BINOMIAL_TEST(Gender, 'M', 0.50, 'TWO_SIDED_PROB'),5) AS Two_Sided_Test
FROM BinomialTest_Table;
```

REAL_PROPORTION	EXACT_TEST	ONE_SIDED_LEFT_TEST	TWO_SIDED_TEST
.41667	.19336	.38721	.58105

and we see that the result does not correspond to our Statistical softwares for example like Minitab:

```
                                                          Valeur
                                                          exacte
Echantillon  X    N  P échantillon        IC à 95 %         de P
1            5   12       0.416667  (0.151652; 0.723330)   0.774
```

or even like **IBM** SPSS (.......):

Binomial Test

		Category	N	Observed Prop.	Test Prop.	Exact Sig. (2-tailed)	Point Probability
Genre	Group 1	M	5	.42	.50	.774	.193
	Group 2	F	7	.58			
	Total		12	1.00			

What happened? It seems that Oracle makes the following mistakes or choice... as you can see below on the Microsoft Excel screenshot:

B15 =SOMME(B2:B7;B10:B14)

	A	B	C	D	E
1	Mens	Probability			
2	0	0.00024414	=LOI.BINOMIALE(A2;12;0.5;0)		
3	1	0.00292969	=LOI.BINOMIALE(A3;12;0.5;0)		
4	2	0.01611328	=LOI.BINOMIALE(A4;12;0.5;0)		
5	3	0.05371094	=LOI.BINOMIALE(A5;12;0.5;0)		
6	4	0.12084961	=LOI.BINOMIALE(A6;12;0.5;0)		
7	5	0.19335938	=LOI.BINOMIALE(A7;12;0.5;0)		
8	6	0.22558594	=LOI.BINOMIALE(A8;12;0.5;0)		
9	7	0.19335938	=LOI.BINOMIALE(A9;12;0.5;0)		
10	8	0.12084961	=LOI.BINOMIALE(A10;12;0.5;0)		
11	9	0.05371094	=LOI.BINOMIALE(A11;12;0.5;0)		
12	10	0.01611328	=LOI.BINOMIALE(A12;12;0.5;0)		
13	11	0.00292969	=LOI.BINOMIALE(A13;12;0.5;0)		
14	12	0.00024414	=LOI.BINOMIALE(A14;12;0.5;0)		
15		0.58105469			
16					

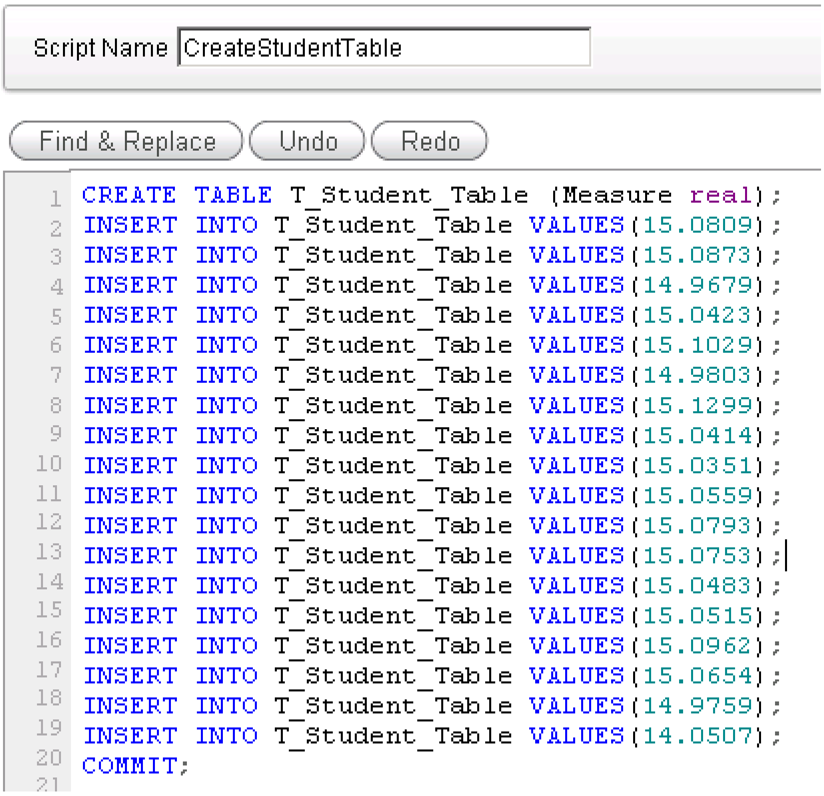
As you can see it does not take the case where Mens=7... to follow

7.1.5.16 SQL Student T-test

7.1.5.16.1 Student One Sample T-test

In the STATS_T_TEST_ONE( ) function the first argument is the sample and the second is the constant mean against which the sample mean is compared. For this t-test only the second argument is optional; the constant mean defaults to 0. This function obtains the value of t by dividing the difference between the sample mean and the known mean by the standard error of the mean.

To see an example, create first a table using the following script:

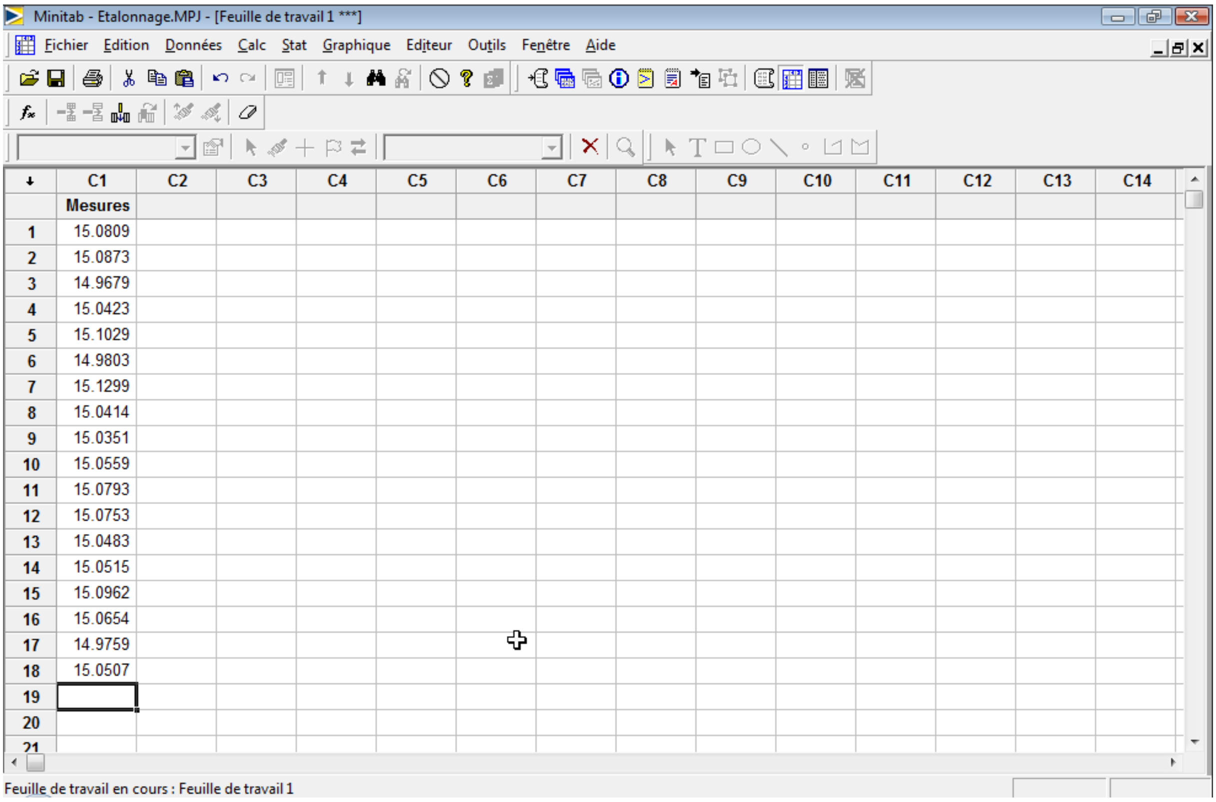


and here is the corresponding text (for copy/paste purpose during the training):

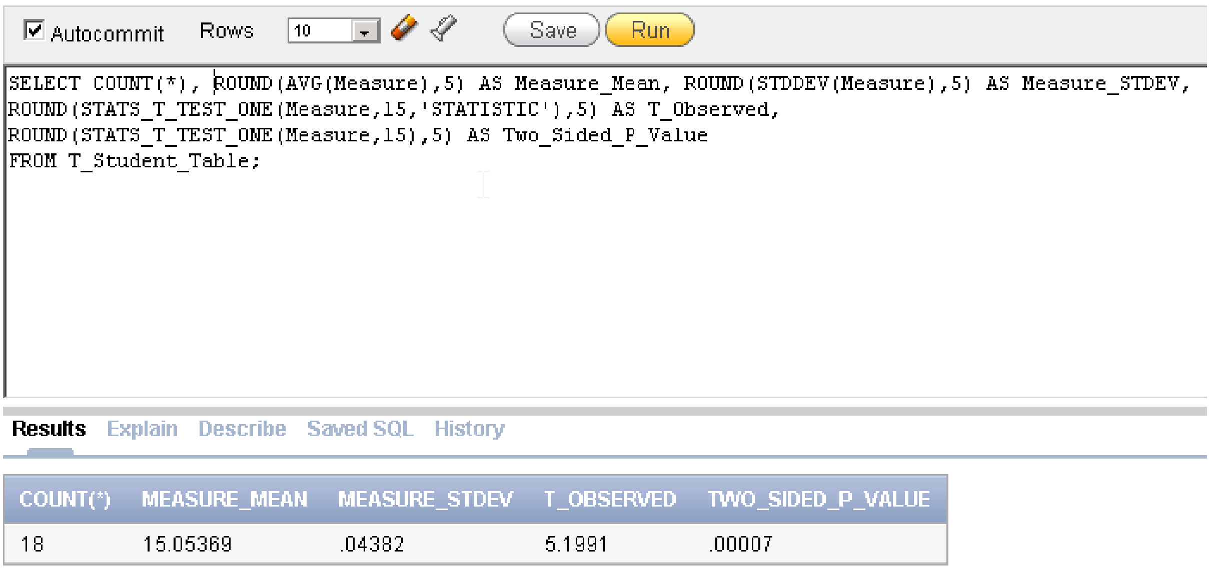
```
CREATE TABLE T_Student_Table (Measure real);
INSERT INTO T_Student_Table VALUES(15.0809);
INSERT INTO T_Student_Table VALUES(15.0873);
INSERT INTO T_Student_Table VALUES(14.9679);
INSERT INTO T_Student_Table VALUES(15.0423);
INSERT INTO T_Student_Table VALUES(15.1029);
INSERT INTO T_Student_Table VALUES(14.9803);
INSERT INTO T_Student_Table VALUES(15.1299);
INSERT INTO T_Student_Table VALUES(15.0414);
INSERT INTO T_Student_Table VALUES(15.0351);
INSERT INTO T_Student_Table VALUES(15.0559);
INSERT INTO T_Student_Table VALUES(15.0793);
INSERT INTO T_Student_Table VALUES(15.0753);
INSERT INTO T_Student_Table VALUES(15.0483);
INSERT INTO T_Student_Table VALUES(15.0515);
```

INSERT INTO T_Student_Table VALUES(15.0962);
INSERT INTO T_Student_Table VALUES(15.0654);
INSERT INTO T_Student_Table VALUES(14.9759);
INSERT INTO T_Student_Table VALUES(15.0507);
COMMIT;

This gives us the table used in Minitab, Tanagra, SPSS and R training:



Now we run the one sample T-Test:



to compare with the result obtained with Minitab during the Statistics course:

Test T à un échantillon : Mesures

```
Test de mu = 15 en fonction de la différence 15


                                          ErT
Variable    N  Moyenne   EcTyp  moyenne        IC à 95 %           T      P
Mesures    18  15.0537  0.0438   0.0103  (15.0319; 15.0755)  5.20  0.000
```

everyithing seems fine ;-)

7.1.5.16.2 Student Two Samples T homoscedastic two-sided test

Before using these functions, it is advisable to determine whether the variances of the samples are significantly different. If they are, then the data may come from distributions with different shapes, and the difference of the means may not be very useful. You can perform an F-test to determine the difference of the variances. If they are not significantly different, use STATS_T_TEST_INDEP( ). If they are significantly different, use STATS_T_TEST_INDEPU( ).

In the STATS_T_TEST_INDEP( ) and STATS_T_TEST_INDEPU( ) functions the first argument is the grouping column and the second is the sample of values.

The following example determines the significance of the difference between the average Pipeline1 and Pipeline2 flow where the distributions are assumed to have similar (pooled) variances:

To do such a test we need to create first a table with the following script:

Script Name CreateStudent2SamplesTable

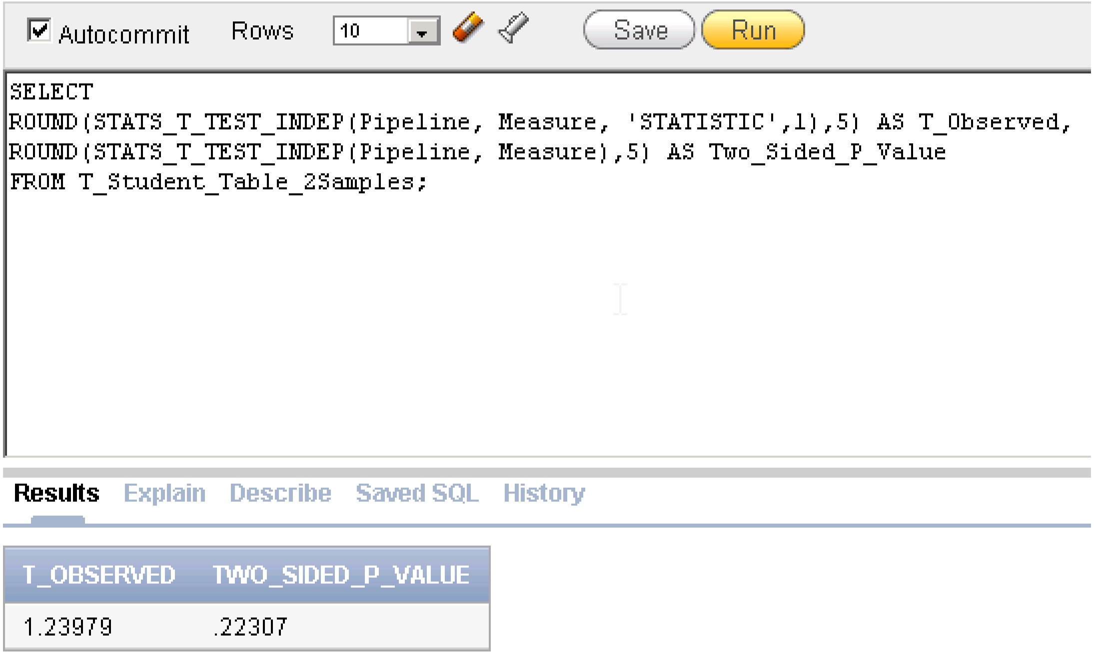
Find & Replace Undo Redo

```
CREATE TABLE T_Student_Table_2Samples (Pipeline integer, Measure integer);
INSERT INTO T_Student_Table_2Samples VALUES(1,163);
INSERT INTO T_Student_Table_2Samples VALUES(1,150);
INSERT INTO T_Student_Table_2Samples VALUES(1,171);
INSERT INTO T_Student_Table_2Samples VALUES(1,155);
INSERT INTO T_Student_Table_2Samples VALUES(1,186);
INSERT INTO T_Student_Table_2Samples VALUES(1,145);
INSERT INTO T_Student_Table_2Samples VALUES(1,154);
INSERT INTO T_Student_Table_2Samples VALUES(1,173);
INSERT INTO T_Student_Table_2Samples VALUES(1,152);
INSERT INTO T_Student_Table_2Samples VALUES(1,150);
INSERT INTO T_Student_Table_2Samples VALUES(1,143);
INSERT INTO T_Student_Table_2Samples VALUES(1,138);
INSERT INTO T_Student_Table_2Samples VALUES(1,166);
INSERT INTO T_Student_Table_2Samples VALUES(1,193);
INSERT INTO T_Student_Table_2Samples VALUES(1,158);
INSERT INTO T_Student_Table_2Samples VALUES(1,175);
INSERT INTO T_Student_Table_2Samples VALUES(1,167);
INSERT INTO T_Student_Table_2Samples VALUES(1,150);
INSERT INTO T_Student_Table_2Samples VALUES(1,158);
INSERT INTO T_Student_Table_2Samples VALUES(2,167);
INSERT INTO T_Student_Table_2Samples VALUES(2,157);
INSERT INTO T_Student_Table_2Samples VALUES(2,149);
INSERT INTO T_Student_Table_2Samples VALUES(2,145);
INSERT INTO T_Student_Table_2Samples VALUES(2,135);
INSERT INTO T_Student_Table_2Samples VALUES(2,157);
INSERT INTO T_Student_Table_2Samples VALUES(2,135);
INSERT INTO T_Student_Table_2Samples VALUES(2,167);
INSERT INTO T_Student_Table_2Samples VALUES(2,154);
INSERT INTO T_Student_Table_2Samples VALUES(2,165);
```

and here is the corresponding full text (for copy/paste purpose during the training):

```
CREATE TABLE T_Student_Table_2Samples (Pipeline integer, Measure integer);
INSERT INTO T_Student_Table_2Samples VALUES(1,163);
INSERT INTO T_Student_Table_2Samples VALUES(1,150);
INSERT INTO T_Student_Table_2Samples VALUES(1,171);
INSERT INTO T_Student_Table_2Samples VALUES(1,155);
INSERT INTO T_Student_Table_2Samples VALUES(1,186);
INSERT INTO T_Student_Table_2Samples VALUES(1,145);
INSERT INTO T_Student_Table_2Samples VALUES(1,154);
INSERT INTO T_Student_Table_2Samples VALUES(1,173);
INSERT INTO T_Student_Table_2Samples VALUES(1,152);
INSERT INTO T_Student_Table_2Samples VALUES(1,150);
INSERT INTO T_Student_Table_2Samples VALUES(1,143);
INSERT INTO T_Student_Table_2Samples VALUES(1,138);
INSERT INTO T_Student_Table_2Samples VALUES(1,166);
INSERT INTO T_Student_Table_2Samples VALUES(1,193);
INSERT INTO T_Student_Table_2Samples VALUES(1,158);
INSERT INTO T_Student_Table_2Samples VALUES(1,175);
INSERT INTO T_Student_Table_2Samples VALUES(1,167);
INSERT INTO T_Student_Table_2Samples VALUES(1,150);
INSERT INTO T_Student_Table_2Samples VALUES(1,158);
INSERT INTO T_Student_Table_2Samples VALUES(2,167);
INSERT INTO T_Student_Table_2Samples VALUES(2,157);
INSERT INTO T_Student_Table_2Samples VALUES(2,149);
INSERT INTO T_Student_Table_2Samples VALUES(2,145);
INSERT INTO T_Student_Table_2Samples VALUES(2,135);
INSERT INTO T_Student_Table_2Samples VALUES(2,157);
INSERT INTO T_Student_Table_2Samples VALUES(2,135);
INSERT INTO T_Student_Table_2Samples VALUES(2,167);
INSERT INTO T_Student_Table_2Samples VALUES(2,154);
INSERT INTO T_Student_Table_2Samples VALUES(2,165);
INSERT INTO T_Student_Table_2Samples VALUES(2,170);
INSERT INTO T_Student_Table_2Samples VALUES(2,165);
INSERT INTO T_Student_Table_2Samples VALUES(2,154);
INSERT INTO T_Student_Table_2Samples VALUES(2,176);
INSERT INTO T_Student_Table_2Samples VALUES(2,155);
INSERT INTO T_Student_Table_2Samples VALUES(2,157);
INSERT INTO T_Student_Table_2Samples VALUES(2,134);
INSERT INTO T_Student_Table_2Samples VALUES(2,156);
INSERT INTO T_Student_Table_2Samples VALUES(2,147);
COMMIT;
```

and then run the following query (the **1** in the third argument specifies which Pipeline is the reference for the calculation!):




```
SELECT
ROUND(STATS_T_TEST_INDEP(Pipeline, Measure, 'STATISTIC',1),5) AS T_Observed,
ROUND(STATS_T_TEST_INDEP(Pipeline, Measure),5) AS Two_Sided_P_Value
FROM T_Student_Table_2Samples;
```

T_OBSERVED	TWO_SIDED_P_VALUE
1.23979	.22307

to compare with the result obtained with Minitab during the Statistics course:

```
Différence = mu (Pipeline1) - mu (Pipeline2)
Estimation de la différence : 5.37
Limites de confiance (à 95 %) pour la différence : (-3.41 ; 14.15)
Test t de la différence = 0 (en fonction de la différence) : Valeur de T = 1.24
     Valeur de p = 0.223  DL = 36
Les deux utilisent l'écart type regroupé = 13.3463
```

everything seems fine ;-)

7.1.5.17 SQL CrossTab Chi-2 test

Typically, cross tabulation (or crosstabs for short) is a statistical process that summarizes categorical data to create a contingency table. They provide a basic picture of the interrelation between two variables and can help find interactions between them.

Because Crosstabs creates a row for each value in one variable and a column for each value in the other, the procedure is not suitable for continuous variables that assume many values.

To see how works regression functions with Oracle we will first create a table with the following script:

Script Name CreateCrossTab_Table

Find & Replace | Undo | Redo

```
CREATE TABLE CrossTab_Table (PrjDelay varchar(1),PrjManager varchar(1));
INSERT INTO CrossTab_Table VALUES('Y','Y');
INSERT INTO CrossTab_Table VALUES('Y','Y');
INSERT INTO CrossTab_Table VALUES('Y','Y');
INSERT INTO CrossTab_Table VALUES('Y','Y');
INSERT INTO CrossTab_Table VALUES('Y','Y');
INSERT INTO CrossTab_Table VALUES('Y','Y');
INSERT INTO CrossTab_Table VALUES('Y','Y');
INSERT INTO CrossTab_Table VALUES('Y','Y');
INSERT INTO CrossTab_Table VALUES('Y','N');
INSERT INTO CrossTab_Table VALUES('N','Y');
INSERT INTO CrossTab_Table VALUES('N','Y');
INSERT INTO CrossTab_Table VALUES('N','Y');
INSERT INTO CrossTab_Table VALUES('N','Y');
INSERT INTO CrossTab_Table VALUES('N','N');
INSERT INTO CrossTab_Table VALUES('N','N');
INSERT INTO CrossTab_Table VALUES('N','N');
INSERT INTO CrossTab_Table VALUES('N','N');
INSERT INTO CrossTab_Table VALUES('N','N');
COMMIT;
```

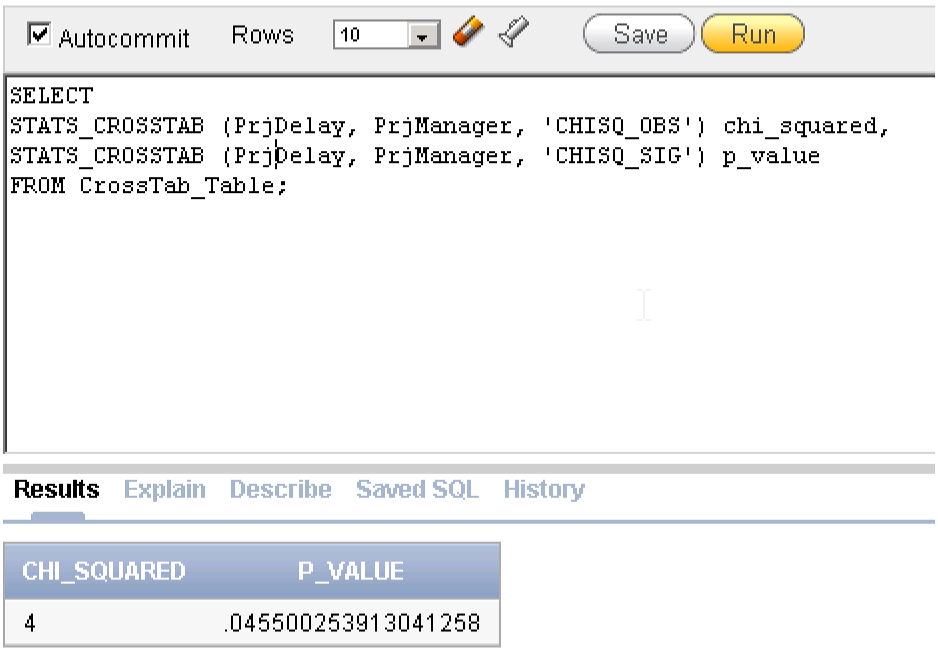
and here is the corresponding full text (for copy/paste purpose during the training):

```
CREATE TABLE CrossTab_Table (PrjDelay varchar(1),PrjManager varchar(1));
INSERT INTO CrossTab_Table VALUES('Y','Y');
INSERT INTO CrossTab_Table VALUES('Y','Y');
INSERT INTO CrossTab_Table VALUES('Y','Y');
INSERT INTO CrossTab_Table VALUES('Y','Y');
INSERT INTO CrossTab_Table VALUES('Y','Y');
INSERT INTO CrossTab_Table VALUES('Y','Y');
INSERT INTO CrossTab_Table VALUES('Y','Y');
INSERT INTO CrossTab_Table VALUES('Y','Y');
INSERT INTO CrossTab_Table VALUES('Y','N');
INSERT INTO CrossTab_Table VALUES('N','Y');
INSERT INTO CrossTab_Table VALUES('N','Y');
INSERT INTO CrossTab_Table VALUES('N','Y');
INSERT INTO CrossTab_Table VALUES('N','Y');
INSERT INTO CrossTab_Table VALUES('N','N');
INSERT INTO CrossTab_Table VALUES('N','N');
INSERT INTO CrossTab_Table VALUES('N','N');
INSERT INTO CrossTab_Table VALUES('N','N');
INSERT INTO CrossTab_Table VALUES('N','N');
COMMIT;
```

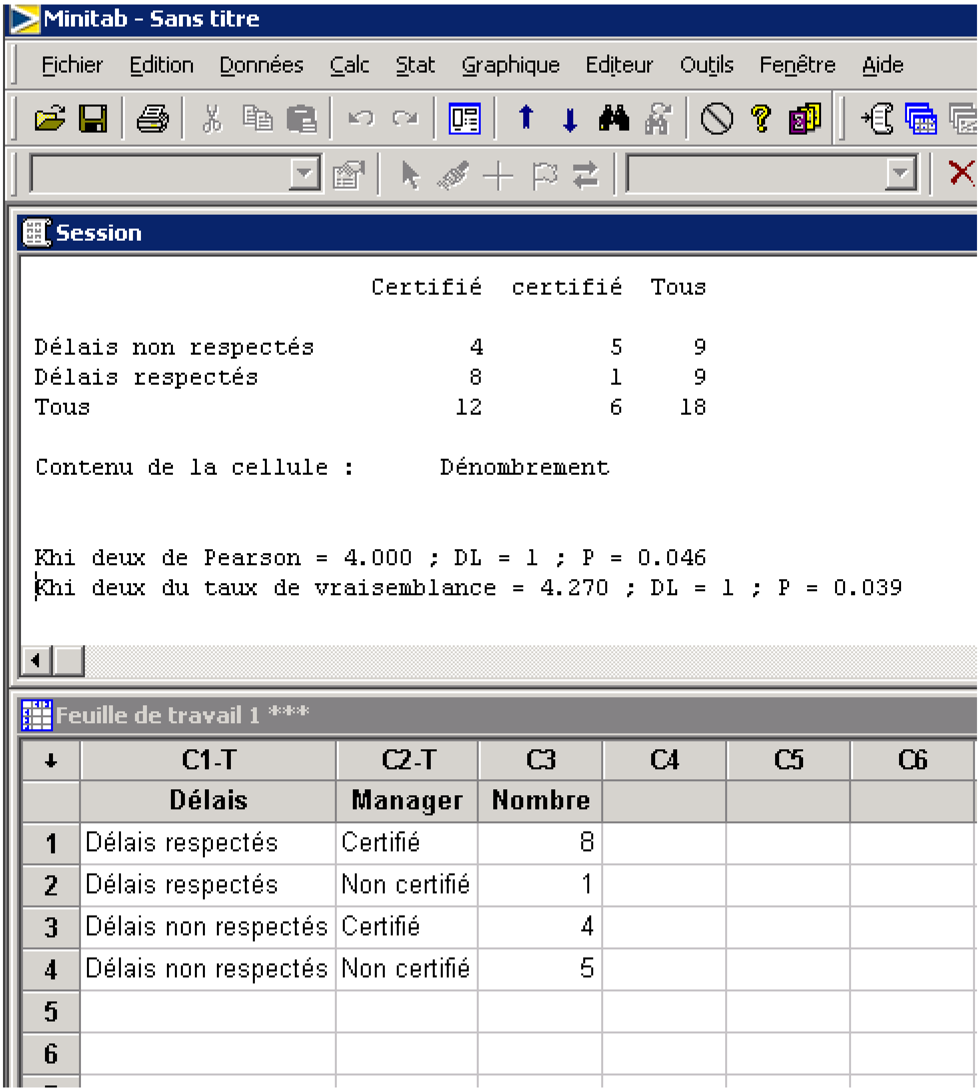
corresponding the following crosstab table corresponding to what we used for the Minitab, Tanagra, SPSS and R training:

Projects	Certified Project Manager	Non-Certified Project Manager	Total
Delays respected	8	1	9
Delays non-respected	4	5	9
Total	12	6	18

And now we run the following query:



And this corresponds perfectly to Minitab output:


Minitab - Sans titre
Fichier Edition Données Calc Stat Graphique Editeur Outils Fenêtre Aide
Session
Certifié certifié Tous
Délais non respectés 4 5 9
Délais respectés 8 1 9
Tous 12 6 18
Contenu de la cellule : Dénombrement
Khi deux de Pearson = 4.000 ; DL = 1 ; P = 0.046
Khi deux du taux de vraisemblance = 4.270 ; DL = 1 ; P = 0.039
Feuille de travail 1 ***
C1-T C2-T C3 C4 C5 C6
Délais Manager Nombre
1 Délais respectés Certifié 8
2 Délais respectés Non certifié 1
3 Délais non respectés Certifié 4
4 Délais non respectés Non certifié 5
5
6

7.1.6 SQL Logical test functions

7.1.6.1 SQL CASE WHEN function

7.1.6.1.1 Inside SELECT Statement

In Oracle and MySQL IF structured is reserved for PL-SQL. You have then to use the CASE WHEN or DECODE statements (caution! The DECODE is considered as depreciated and should not be used anymore, furthermore it is ORACLE specific).

The CASE ... WHEN statement can be used for multiple IF simplifications. Here an example with the W3School database:

```
SELECT CustomerName, Country,
    CASE Country
        WHEN 'Germany' THEN '2 days shipping delay'
        WHEN 'Mexico' THEN '20 days shipping delay'
        ELSE '15 days shipping delay'
    END AS ShippingDelay
FROM
    Customers;
```

That will result in:

CustomerName	Country	ShippingDelay
Alfreds Futterkiste	Germany	2 days shipping delay
Ana Trujillo Emparedados y helados	Mexico	20 days shipping delay
Antonio Moreno Taquería	Mexico	20 days shipping delay
Around the Horn	UK	15 days shipping delay
Berglunds snabbköp	Sweden	15 days shipping delay
Blauer See Delikatessen	Germany	2 days shipping delay
Blondel père et fils	France	15 days shipping delay
Bólido Comidas preparadas	Spain	15 days shipping delay
Bon app'	France	15 days shipping delay
Bottom-Dollar Marketse	Canada	15 days shipping delay
B's Beverages	UK	15 days shipping delay
.....		

Or consider the more complete example with Oracle mixing different tables and SQL statements and functions:

With the following tables:

DEMO_CUSTOMERS

Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependencies SQL

Query Count Rows Insert Row

EDIT	CUSTOMER_ID	CUST_FIRST_NAME	CUST_LAST_NAME	CUST_STREET_ADDRESS1	CUST_STREET_ADDRESS2	CUST_CITY	CUST_STATE	CUST_POSTAL_CODE	PHONE_NUMBER1
	1	John	Dulles	45020 Aviation Drive	-	Sterling	VA	20166	703-555-2143
	2	William	Hartsfield	6000 North Terminal Parkway	-	Atlanta	GA	30320	404-555-3285
	3	Edward	Logan	1 Harborside Drive	-	East Boston	MA	02128	617-555-3295
	4	Edward "Butch"	OHare	10000 West OHare	-	Chicago	IL	60666	773-555-7693
	5	Fiorello	LaGuardia	Hangar Center	Third Floor	Flushing	NY	11371	212-555-3923
	6	Albert	Lambert	10701 Lambert International Blvd.	-	St. Louis	MO	63145	314-555-4022

DEMO_ORDERS

Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependencies SQL

Query Count Rows Insert Row

EDIT	ORDER_ID	CUSTOMER_ID	ORDER_TOTAL	ORDER_TIMESTAMP	USER_ID
	1	7	1890	09/02/2013	2
	2	1	2380	08/30/2013	2
	3	2	1640	08/24/2013	2
	4	5	1090	08/16/2013	2
	5	6	950	08/11/2013	2
	6	3	1515	08/06/2013	2
	7	3	905	07/27/2013	2
	8	4	1060	07/25/2013	2
	9	2	730	07/14/2013	2
	10	7	870	06/30/2013	2

row(s) 1 - 10 of 10

DEMO_ORDER_ITEMS

Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependencies SQL

Query Count Rows Insert Row

EDIT	ORDER_ITEM_ID	ORDER_ID	PRODUCT_ID	UNIT_PRICE	QUANTITY
	1	1	1	50	10
	2	1	2	80	8
	3	1	3	150	5
	4	2	1	50	3
	5	2	2	80	3
	6	2	3	150	3
	7	2	4	60	3
	8	2	5	80	3
	9	2	6	120	2
	10	2	7	30	2
	11	2	8	125	4

DEMO_PRODUCT_INFO

Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependencies SQL

Query | Count Rows | Insert Row

EDIT	PRODUCT_ID	PRODUCT_NAME	PRODUCT_DESCRIPTION	CATEGORY	PRODUCT_AVAIL	LIST_PRICE
	1	Business Shirt	Wrinkle-free cotton business shirt	Mens	Y	50
	2	Trousers	Black trousers suitable for every business man	Mens	Y	80
	3	Jacket	Fully lined jacket which is both professional and extremely comfortable to wear	Mens	Y	150
	4	Blouse	Silk blouse ideal for all business women	Womens	Y	60
	5	Skirt	Wrinkle free skirt	Womens	Y	80
	6	Ladies Shoes	Low heel and cushioned interior for comfort and style in simple yet elegant shoes	Womens	Y	120
	7	Belt	Leather belt	Accessories	Y	30

DEMO_STATES

Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependencies SQL

Query | Count Rows | Insert Row

EDIT	ST	STATE_NAME
	AK	ALASKA
	AL	ALABAMA
	AR	ARKANSAS
	AZ	ARIZONA
	CA	CALIFORNIA
	CO	COLORADO
	CT	CONNECTICUT
	DC	DISTRICT OF COLUMBIA
	DE	DELAWARE
	FL	FLORIDA
	GA	GEORGIA

```
SELECT state,
       sum(mens) "Mens",
       sum(womens) "Womens",
       sum(accessories) "Accessories"
FROM (SELECT demo_customers.cust_state state,
                CASE
                   WHEN demo_product_info.category = 'Mens'
                   THEN
                      demo_order_items.quantity *
demo_order_items.unit_price
                   ELSE
                      0
                END
                   mens,
                CASE
                   WHEN demo_product_info.category = 'Womens'
                   THEN
                      demo_order_items.quantity *
demo_order_items.unit_price
                   ELSE
                      0
                END
                   womens,
                CASE
                   WHEN demo_product_info.category = 'Accessories'
                   THEN
                      demo_order_items.quantity *
demo_order_items.unit_price
                   ELSE
                      0
                END
                   accessories
         FROM   demo_order_items,
                demo_product_info,
                demo_customers,
                demo_orders
         WHERE  demo_order_items.product_id = demo_product_info.product_id
                AND demo_order_items.order_id = demo_orders.order_id
                AND demo_orders.customer_id = demo_customers.customer_id )
GROUP BY ROLLUP( state );
```

This SQL query will result in:

Results Explain Describe Saved SQL History

STATE	Mens	Womens	Accessories
CT	2760	0	0
GA	0	1520	850
IL	940	120	0
MA	1040	640	740
MO	610	340	0
NY	220	240	630
VA	1060	660	660
-	6630	3520	2880

Case versus Decode:

CASE:	DECODE:
• Complies ANSI SQL	• Oracle Proprietary
• Can work with logical operators other than '='	• Works with only '=' / like operator
• Can work with predicated and searchable queries	• Expressions are scalar values only
• Needs data consistency	• Data consistency is not needed
• NULL=NULL returns FALSE	• NULL IS NULL returns TRUE
• Can be used in PL/SQL and SQL statements	• Can be used in SQL Statemens
• Can be used in parameters while calling a procedure	• Cannot be used in parameters while calling a procedure

Here is an example that illustrated a typical difference:

```
SELECT ename, empno,
DECODE( deptno ,10 ,'Accounting'
               ,20 ,'Research'
               ,30 ,'Sales'
               ,40 ,'Operations'
               ,'Unknown'
) department
FROM emp
ORDER BY ename;
```

```
SELECT
 (CASE
WHEN sal < 1000 THEN 'Low'
WHEN sal BETWEEN 1000 AND 3000 THEN 'Medium'
WHEN sal > 3000 THEN 'High'
ELSE 'N/A'
 END) salary
FROM emp
ORDER BY ename;
```

And here is an example of CASE using the IN predicates ::

```
SELECT CASE
-- predicate with "IN"
WHEN salary IN (9000,10000) THEN '9K-10K'
----searchable subquery
WHEN EMP_NO IN (SELECT mgr_no FROM department) THEN 'Dept_Mgr'
ELSE 'Unknown'
END category
   FROM employee ;
```

And other exemple that highlights the fact that one cares about consistency and the other not:

```
SELECT DECODE(200,100,100,'200','200','300') TEST
FROM dual;

--------Output:

TEST
----
200
```

```
SELECT CASE 200 WHEN 100 THEN 100
 WHEN '200' THEN '200'
 ELSE '300'
END TEST
FROM dual;
------------
Error on line 2 at position 14 WHEN '200' THEN '200'
ORA-00932: inconsistent datatypes: expected NUMBER got CHAR
```

7.1.6.1.2 Inside WHERE Statement

An another very interesting application of WHEN CASE is to use it in an ORDER BY statement as you can see below:

Autocommit Rows 10 Save Run

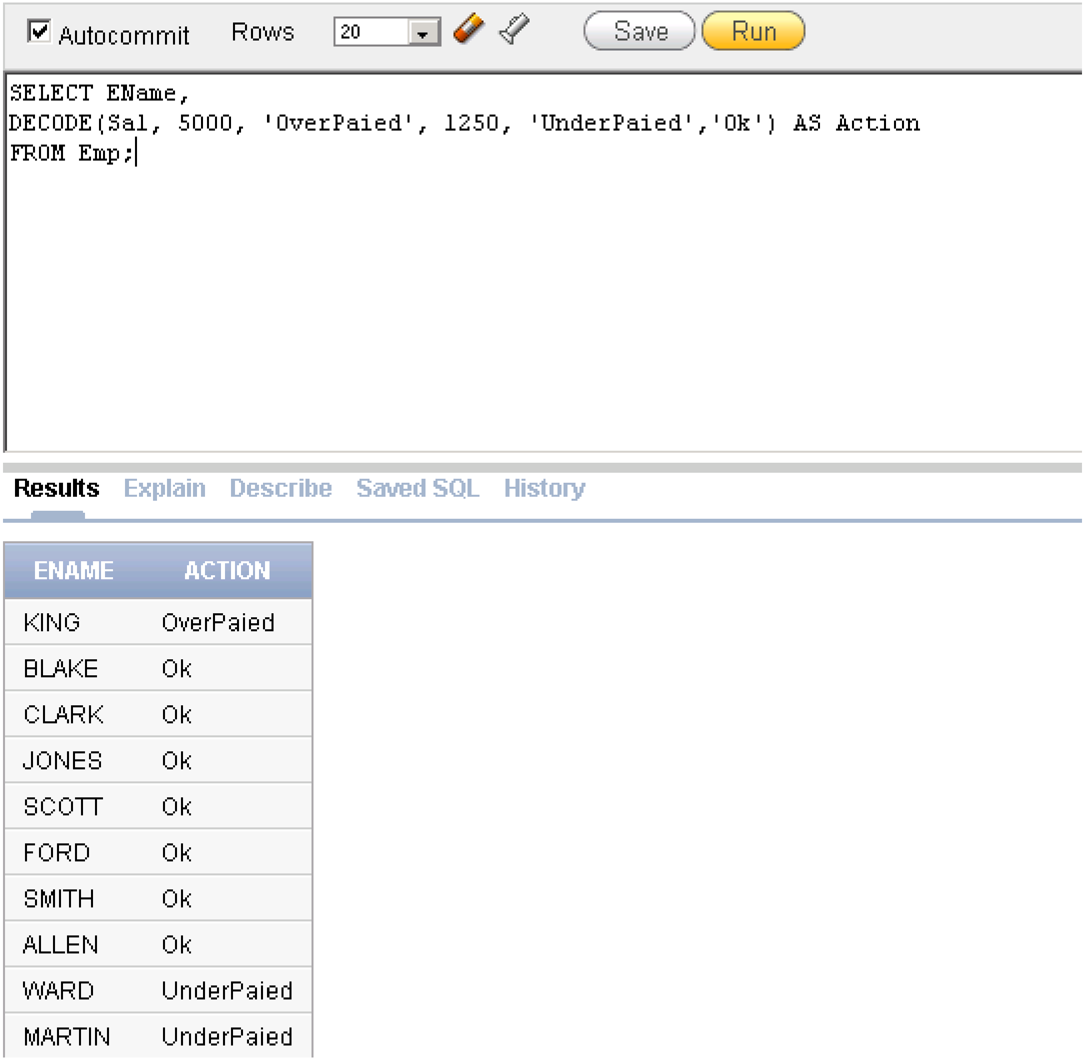
```
SELECT Ename, Sal, Job, Comm
FROM Emp
ORDER BY CASE WHEN JOB  = 'SALESMAN' THEN COMM ELSE Sal END;
```

Results Explain Describe Saved SQL History

ENAME	SAL	JOB	COMM
TURNER	1807.21	SALESMAN	0
ALLEN	1927.69	SALESMAN	300
WARD	1506.01	SALESMAN	500
SMITH	800	CLERK	-
ADAMS	1100	CLERK	-
JAMES	1144.56	CLERK	-
MILLER	1300	CLERK	-
MARTIN	1506.01	SALESMAN	1400
CLARK	2450	MANAGER	-
JONES	2975	MANAGER	-

7.1.6.2 SQL DECODE function:

In Oracle/PLSQL, the DECODE function has the functionality of an IF-THEN-ELSE statement or of a CASE statement **but without comparison operators as already mentioned**!!!




```
SELECT EName,
DECODE(Sal, 5000, 'OverPaied', 1250, 'UnderPaied','Ok') AS Action
FROM Emp;
```

ENAME	ACTION
KING	OverPaied
BLAKE	Ok
CLARK	Ok
JONES	Ok
SCOTT	Ok
FORD	Ok
SMITH	Ok
ALLEN	Ok
WARD	UnderPaied
MARTIN	UnderPaied

7.1.6.3 SQL MERGE INTO USING... MATCHED::

The purpose of SQL MERGE INTO in association with WHEN is to avoid creating multiple queries to update datas for example.

To study MERGE INTO we will use first de default EMP table available in Oracle:

EMP

Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependencies SQL

Query Count Rows Insert Row

EDIT	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
	7839	KING	PRESIDENT	-	11/17/1981	5000	-	10
	7698	BLAKE	MANAGER	7839	05/01/1981	2850	-	30
	7782	CLARK	MANAGER	7839	06/09/1981	2450	-	10
	7566	JONES	MANAGER	7839	04/02/1981	2975	-	20
	7788	SCOTT	ANALYST	7566	12/09/1982	3000	-	20
	7902	FORD	ANALYST	7566	12/03/1981	3000	-	20
	7369	SMITH	CLERK	7902	12/17/1980	800	-	20
	7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
	7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
	7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
	7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
	7876	ADAMS	CLERK	7788	01/12/1983	1100	-	20

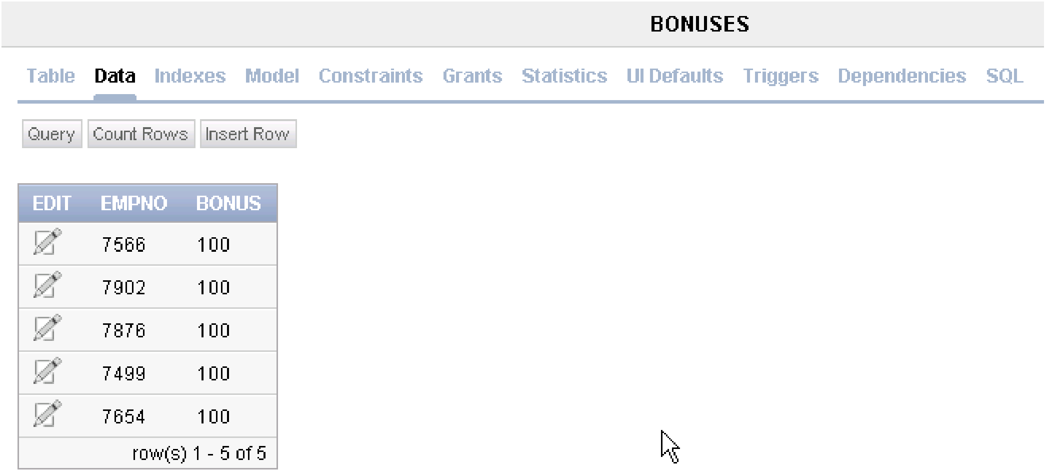
and create also a Bonus table (now using a script instead of INSERT ALL for fun):

Script Name CreateBonusTable Cancel Download Delete Save Run

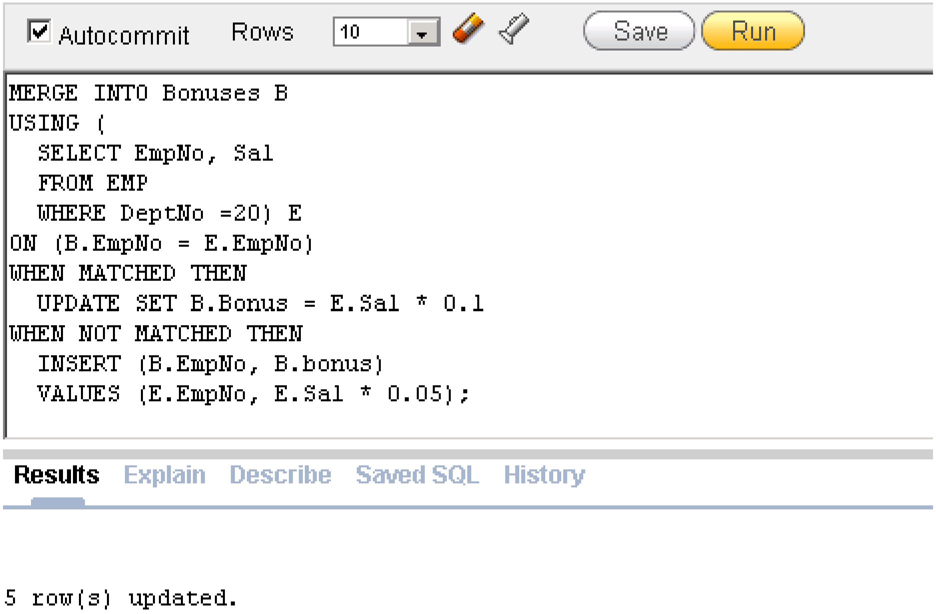
Find & Replace Undo Redo

```
CREATE TABLE Bonuses (
EmpNo NUMBER, bonus NUMBER DEFAULT 100);

INSERT INTO Bonuses (EmpNo) VALUES (7566);
INSERT INTO Bonuses (EmpNo) VALUES (7902);
INSERT INTO Bonuses (EmpNo) VALUES (7876);
INSERT INTO Bonuses (EmpNo) VALUES (7499);
INSERT INTO Bonuses (EmpNo) VALUES (7654);
COMMIT;
```

this will give:



And now here is the MERGE INTO example:



The result will be:

EDIT	EMPNO	BONUS
	7566	297.5
	7902	300
	7876	110
	7499	100
	7654	100
	7369	40
	7788	150
		row(s) 1 - 7 of 7

As you can see there a two more rows and the existing one have the bonus updated.

7.1.7 SQL Text functions

7.1.7.1 SQL UCASE/LCASE function

The UCASE()/LCASE() functions converts the value of a field to uppercase/lowercase.

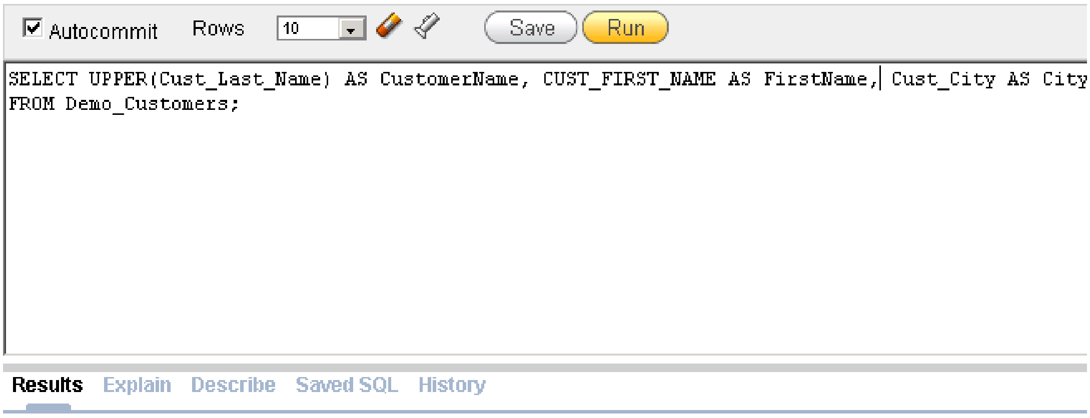
SQL UCASE()/LCASE() Syntax:

```
SELECT UCASE(column_name) FROM table_name;
```

Syntax for SQL Server and ORACLE using UPPER()/LOWER():

```
SELECT UPPER(column_name) FROM table_name;
```

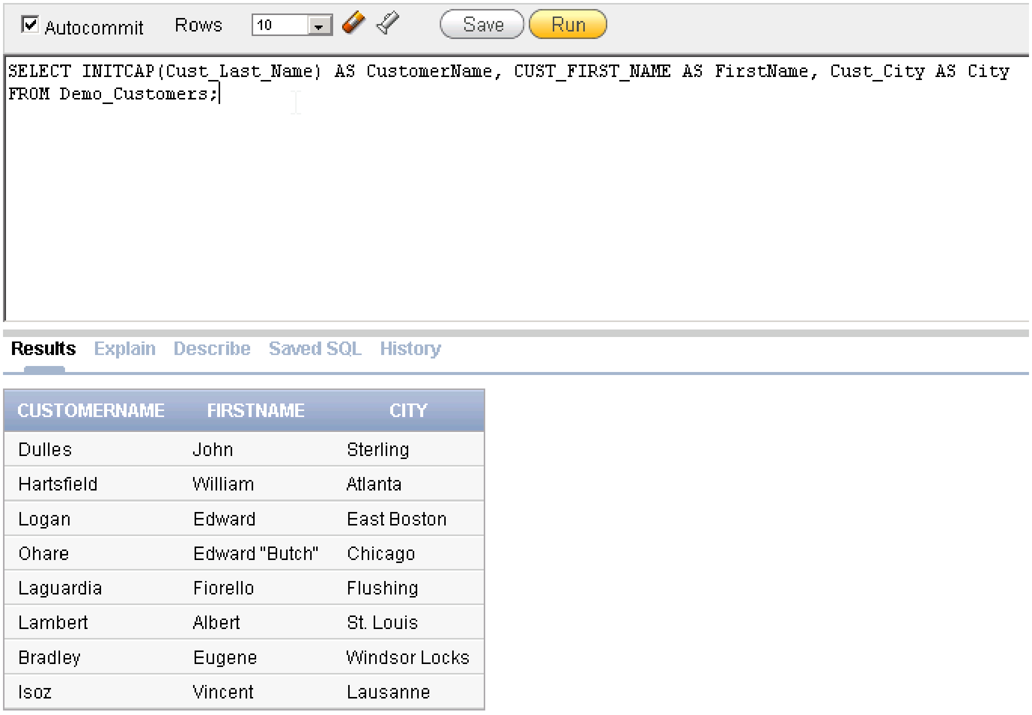
Example with Oracle:



CUSTOMERNAME	FIRSTNAME	CITY
DULLES	John	Sterling
HARTSFIELD	William	Atlanta
LOGAN	Edward	East Boston
OHARE	Edward "Butch"	Chicago
LAGUARDIA	Fiorello	Flushing
LAMBERT	Albert	St. Louis
BRADLEY	Eugene	Windsor Locks
ISOZ	Vincent	Lausanne

7.1.7.2 SQL INITCAP function

In Oracle/PLSQL, the INITCAP( ) function sets the first character in each word to uppercase and the rest to lowercase.




```
SELECT INITCAP(Cust_Last_Name) AS CustomerName, CUST_FIRST_NAME AS FirstName, Cust_City AS City
FROM Demo_Customers;
```

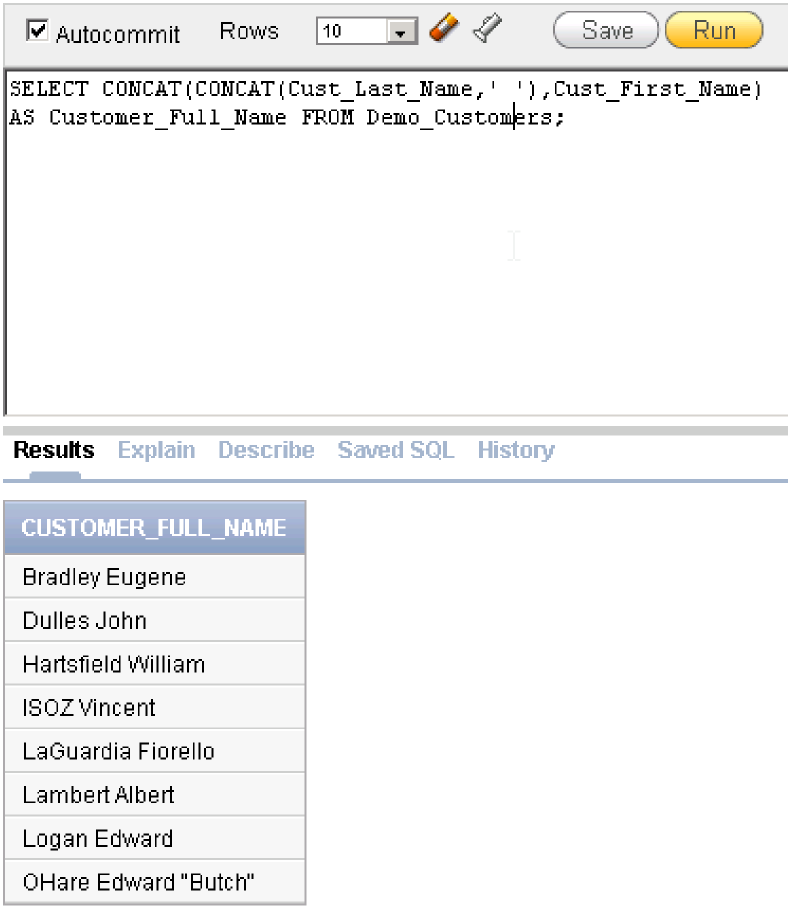
CUSTOMERNAME	FIRSTNAME	CITY
Dulles	John	Sterling
Hartsfield	William	Atlanta
Logan	Edward	East Boston
Ohare	Edward "Butch"	Chicago
Laguardia	Fiorello	Flushing
Lambert	Albert	St. Louis
Bradley	Eugene	Windsor Locks
Isoz	Vincent	Lausanne

7.1.7.3 SQL Concatenate function

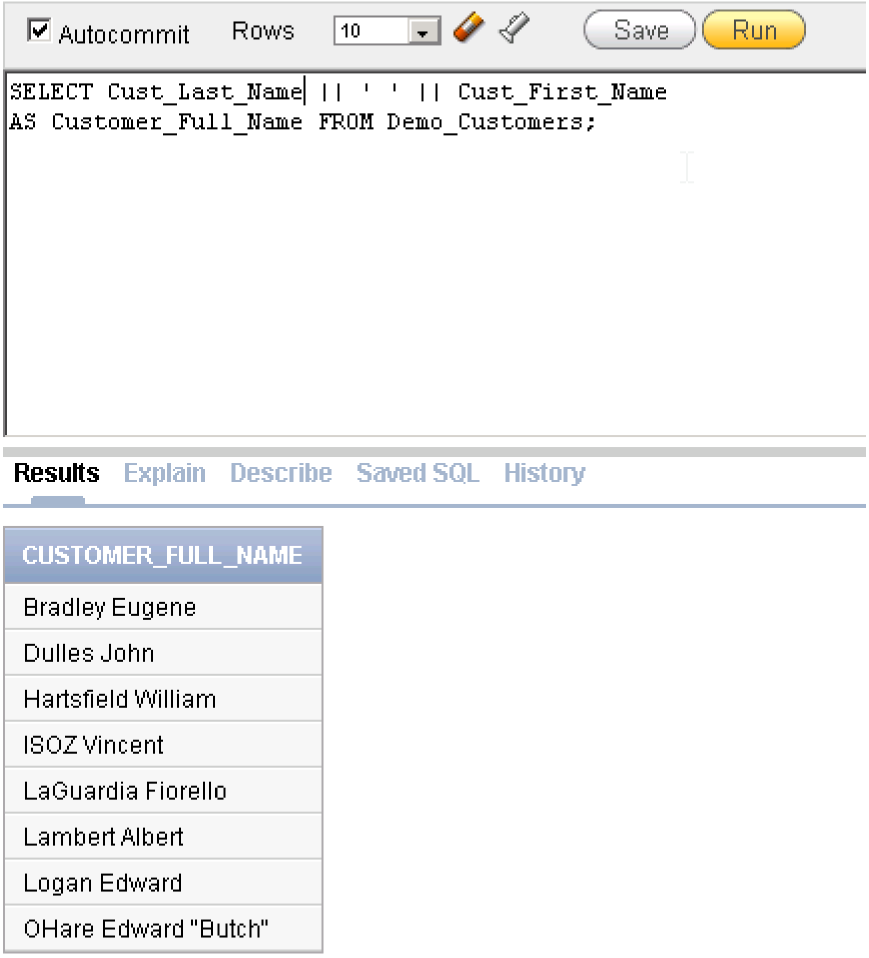
It is sometimes necessary to combine together (concatenate) the results of several different fields. Each database has its own method of concatenation (...):

- MySQL: CONCAT( )
- Oracle: CONCAT( ) or ||
- SQL Server: +

In Oracle CONCAT takes only two arguments. Then if you need three you have at least two choices:



Or:


Autocommit
Rows
10
Save
Run
SELECT Cust_Last_Name || ' ' || Cust_First_Name
AS Customer_Full_Name FROM Demo_Customers;
Results
Explain
Describe
Saved SQL
History
CUSTOMER_FULL_NAME
Bradley Eugene
Dulles John
Hartsfield William
ISOZ Vincent
LaGuardia Fiorello
Lambert Albert
Logan Edward
OHare Edward "Butch"

7.1.7.4 SQL SUBSTRING (MID) function

The MID() function is used to extract characters from a text field.

SQL MID() Syntax in MySQL and MS Access:

```
SELECT MID(column_name,start[,length]) FROM table_name;
```

Oracle doesn't have some of the handy short-hand functions that Microsoft has embedded into it's VB programming languages and into SQL Server but, of course, provides a similar way to return the same result.

The key, is Oracle's SUBSTR( ) function!

In Microsoft's SQL Server, and in Visual Basic, you have the following:

MID(YourStringHere, StartFrom, NumCharsToGrab)
MID("birthday",1,5) = "birth"
MID("birthday",5,2) = "hd"

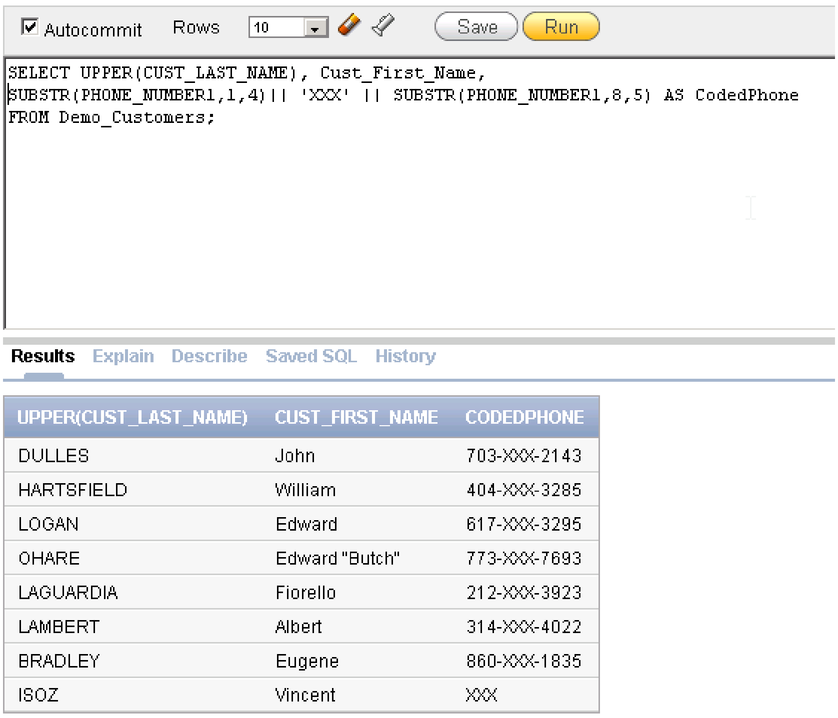
LEFT(YourStringHere,NumCharsToGrab)
LEFT("birthday",5) = "birth"
LEFT("birthday",1) = "b"

RIGHT(YourStringHere,NumCharsToGrab)
RIGHT("birthday",3) = "day"
RIGHT("birthday",1) = "y"

Oracle's SUBSTR function works much the same as the MID function:

SUBSTR(YourStringHere,StartFrom,NumCharsToGrab)
SUBSTR("birthday",1,2) = "bi"
SUBSTR("birthday",-2,2) = "ay"   the -2 indicates started from the end of the word

Here is an example with Oracle:



UPPER(CUST_LAST_NAME)	CUST_FIRST_NAME	CODEDPHONE
DULLES	John	703-XXX-2143
HARTSFIELD	William	404-XXX-3285
LOGAN	Edward	617-XXX-3295
OHARE	Edward "Butch"	773-XXX-7693
LAGUARDIA	Fiorello	212-XXX-3923
LAMBERT	Albert	314-XXX-4022
BRADLEY	Eugene	860-XXX-1835
ISOZ	Vincent	XXX

or a little bit more elaborated:

Autocommit Rows 10 Save Run

```
SELECT Cust_Last_Name,
    CASE NVL(Phone_Number1,'null')
        WHEN 'null' THEN 'NA'
        ELSE SUBSTR(PHONE_NUMBER1,1,4) || 'XXX' || SUBSTR(PHONE_NUMBER1,8,5)
    END AS CodedPhone
FROM
    Demo_Customers;
```

Results Explain Describe Saved SQL History

CUST_LAST_NAME	CODEDPHONE
Dulles	703-XXX-2143
Hartsfield	404-XXX-3285
Logan	617-XXX-3295
OHare	773-XXX-7693
LaGuardia	212-XXX-3923
Lambert	314-XXX-4022
Bradley	860-XXX-1835
ISOZ	NA

7.1.7.5 SQL LEN function

The LEN( ) function returns the length of the value in a text field.

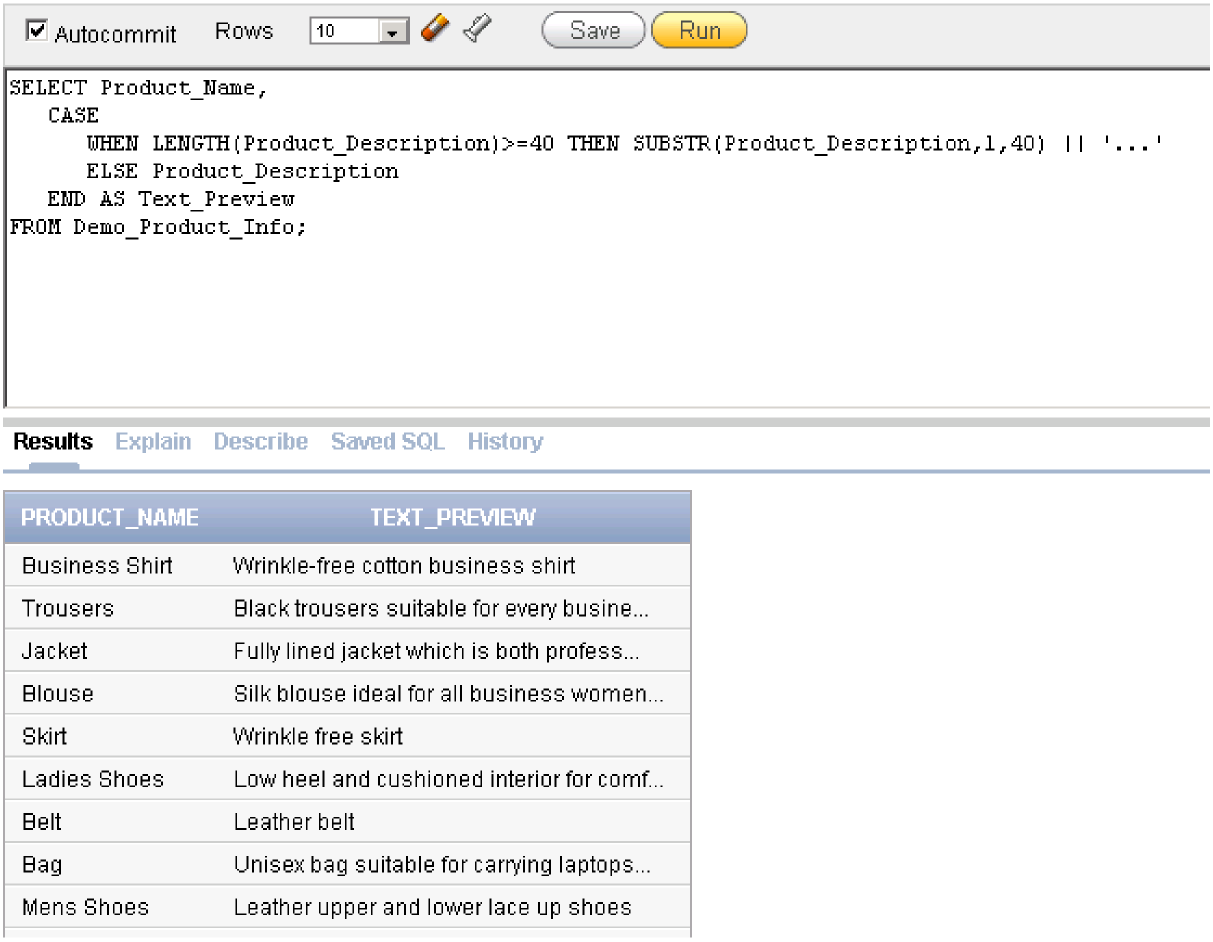
SQL LEN( ) Syntax for mySQL and MS Access:

```
SELECT LEN(column_name) FROM table_name;
```

SQL LENGTH() Syntax for Oracle:

```
SELECT LENGTH(column_name) FROM table_name;
```

Here is a typical example used a lot on the web:



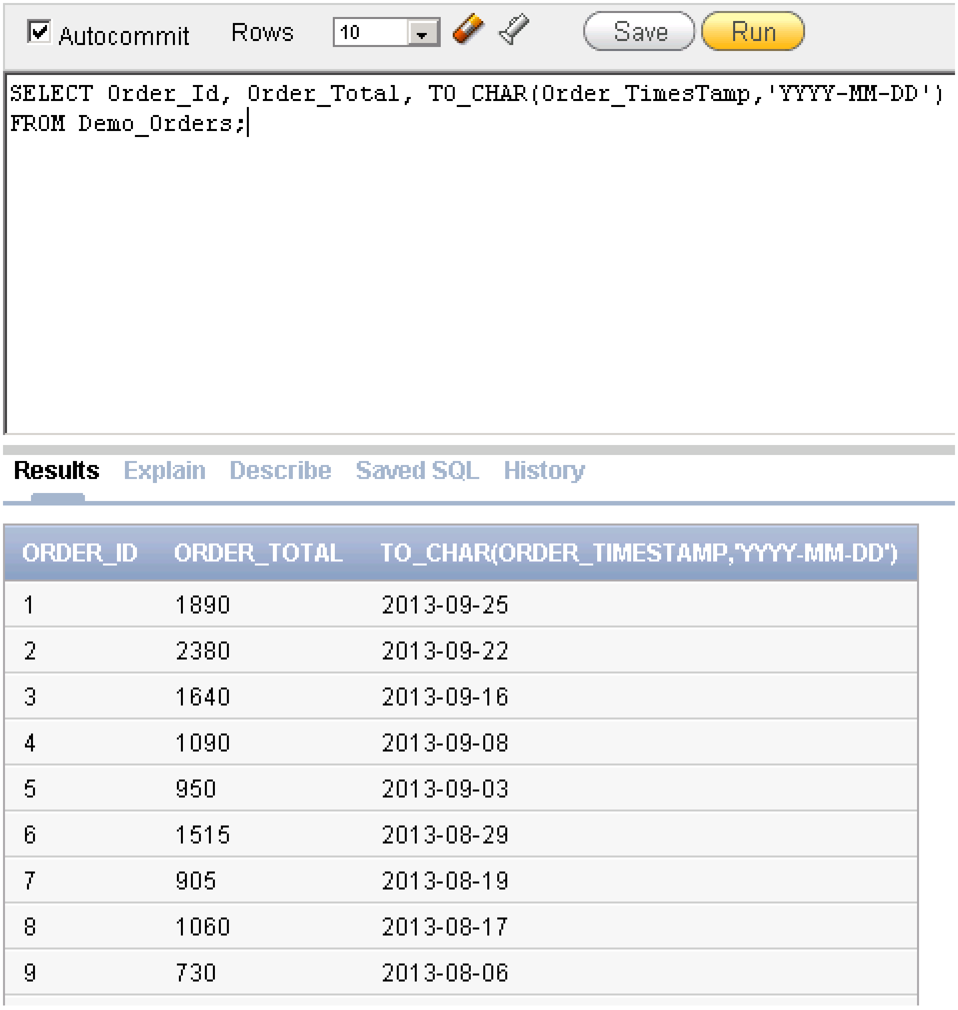
7.1.7.6 SQL format text function (TO_CHAR)

In Oracle/PLSQL, the TO_CHAR( ) function converts a number or date to a string.

The syntax for the TO_CHAR( ) function is:

```
TO_CHAR( value, [ format_mask ], [ nls_language ] )
```

Here is a typical first example:



and a second well know example (already seen before):

```
SELECT TO_CHAR(Order_TimeStamp,'IW') AS ISO_Week_Nb,
SUM(Unit_Price*Quantity) AS Sum, ROUND(AVG(Unit_Price*Quantity),2) AS Average,
MEDIAN(Unit_Price*Quantity) AS Median,
ROUND(STDDEV(Unit_Price*Quantity),2) AS Standard_Deviation,
ROUND(VARIANCE(Unit_Price*Quantity),2) AS Variance
FROM Demo_Orders
INNER JOIN Demo_Order_Items
ON Demo_Orders.Order_Id=Demo_Order_Items.Order_Id
GROUP BY TO_CHAR(Order_TimeStamp,'IW');
```

Results Explain Describe Saved SQL History

ISO_WEEK_NB	SUM	AVERAGE	MEDIAN	STANDARD_DEVIATION	VARIANCE
30	870	290	300	36.06	1300
32	730	243.33	240	5.77	33.33
33	1060	265	245	153.3	23500
34	905	129.29	125	28.35	803.57
35	1515	378.75	367.5	51.05	2606.25
36	2040	204	190	48.12	2315.56
38	4020	268	240	150.06	22517.14
39	1890	630	640	125.3	15700

and the same again with **number formatting** (**that will cause in Microsoft Excel the numbers to be in text format!!!!**) to obtain thousand separators using American representation:

Autocommit Rows 10 Save Run

```
SELECT TO_CHAR(Order_TimeStamp,'IW') AS ISO_Week_Nb,
TO_CHAR(SUM(Unit_Price*Quantity),'9,999.99') AS Sum, ROUND(AVG(Unit_Price*Quantity),2) AS Average,
MEDIAN(Unit_Price*Quantity) AS Median,
ROUND(STDDEV(Unit_Price*Quantity),2) AS Standard_Deviation,
TO_CHAR(ROUND(VARIANCE(Unit_Price*Quantity),2),'999,999.99') AS Variance
FROM Demo_Orders
INNER JOIN Demo_Order_Items
ON Demo_Orders.Order_Id=Demo_Order_Items.Order_Id
GROUP BY TO_CHAR(Order_TimeStamp,'IW');
```

Results Explain Describe Saved SQL History

ISO_WEEK_NB	SUM	AVERAGE	MEDIAN	STANDARD_DEVIATION	VARIANCE
30	870.00	290	300	36.06	1,300.00
32	730.00	243.33	240	5.77	33.33
33	1,060.00	265	245	153.3	23,500.00
34	905.00	129.29	125	28.35	803.57
35	1,515.00	378.75	367.5	51.05	2,606.25
36	2,040.00	204	190	48.12	2,315.56
38	4,020.00	268	240	150.06	22,517.14
39	1,890.00	630	640	125.3	15,700.00

7.1.7.7 SQL REPLACE function

In Oracle/PLSQL, the REPLACE( ) function replaces a sequence of characters in a string with another set of characters.

The syntax for the REPLACE( ) function is:

```
REPLACE( string1, string_to_replace, [ replacement_string ] )
```

If we take the previous example but in the main to obtain thousand seperator using European representation, we get:

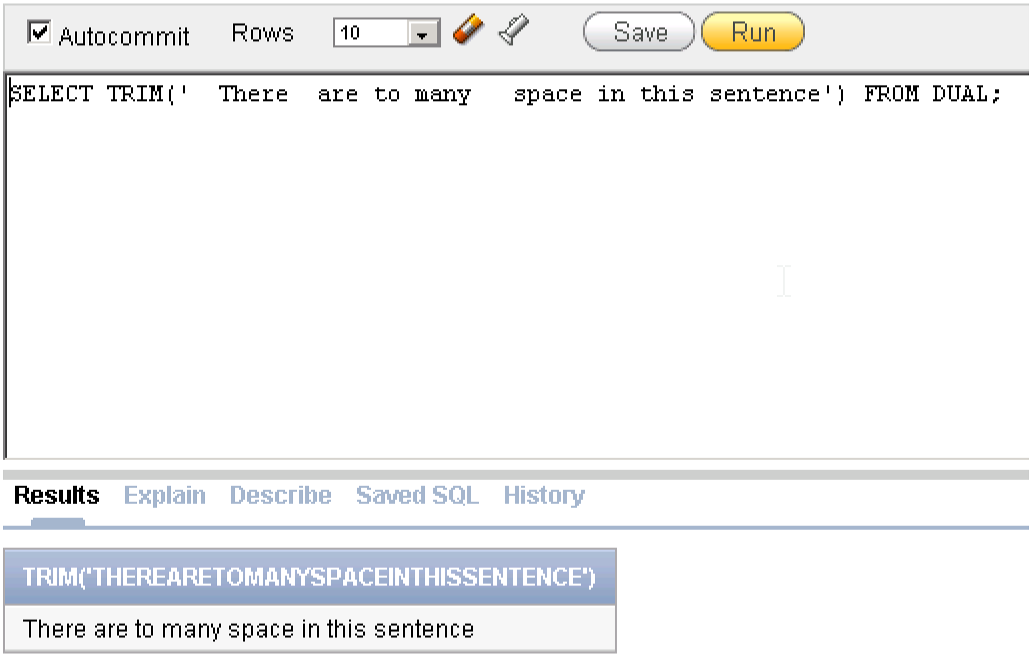
```
SELECT TO_CHAR(Order_TimeStamp,'IW') AS ISO_Week_Nb,
REPLACE(TO_CHAR(SUM(Unit_Price*Quantity),'9,999.99'),',','''') AS Sum, ROUND(AVG(Unit_Price*Quantity),2) AS Average,
MEDIAN(Unit_Price*Quantity) AS Median,
ROUND(STDDEV(Unit_Price*Quantity),2) AS Standard_Deviation,
REPLACE(TO_CHAR(ROUND(VARIANCE(Unit_Price*Quantity),2),'999,999.99'),',','''') AS Variance
FROM Demo_Orders
INNER JOIN Demo_Order_Items
ON Demo_Orders.Order_Id=Demo_Order_Items.Order_Id
GROUP BY TO_CHAR(Order_TimeStamp,'IW');
```

ISO_WEEK_NB	SUM	AVERAGE	MEDIAN	STANDARD_DEVIATION	VARIANCE
30	870.00	290	300	36.06	1'300.00
32	730.00	243.33	240	5.77	33.33
33	1'060.00	265	245	153.3	23'500.00
34	905.00	129.29	125	28.35	803.57
35	1'515.00	378.75	367.5	51.05	2'606.25
36	2'040.00	204	190	48.12	2'315.56
38	4'020.00	268	240	150.06	22'517.14
39	1'890.00	630	640	125.3	15'700.00

7.1.7.8 SQL TRIM function

The TRIM( ) function in mainly used for Internet because users type sometimes a lot of useless blank spaces anywhere in the input. Then an easy way to clean all duplicates spaces is to use TRIM( ).

A generic example must be enough to understand how it works:



7.1.7.9 SQL LPAD function

In Oracle/PLSQL, the LPAD( ) function pads the left-side of a string with a specific set of characters

Remember the chapter about CONNECT BY of page 71 with the following orghchart ;-)

Autocommit Rows 20 Save Run

```
SELECT LPAD(E.EName,LENGTH(E.EName)+(LEVEL-1)*3,'+') "Horizontal Orgchart"
FROM Emp E
WHERE E.EName <> 'BLAKE'
START WITH E.Mgr IS NULL
CONNECT BY E.Mgr = PRIOR E.EmpNo;
```

Results Explain Describe Saved SQL History

Horizontal Orgchart
KING
+++JONES
++++++SCOTT
+++++++++ADAMS
++++++FORD
+++++++++SMITH
++++++ALLEN
++++++WARD
++++++MARTIN
++++++TURNER
++++++JAMES
+++CLARK
++++++MILLER

7.1.8 SQL Dates functions

7.1.8.1 SQL Now function

The SYSDATE( ) function (also NOW() on some RDMS) is used a lot as default value when we create columns. But it's also used a lot to calculate the number of days, month, years... between a date in a table and... now (typically in project management applications).

Here is a typical example:

Autocommit Rows 10 Save Run

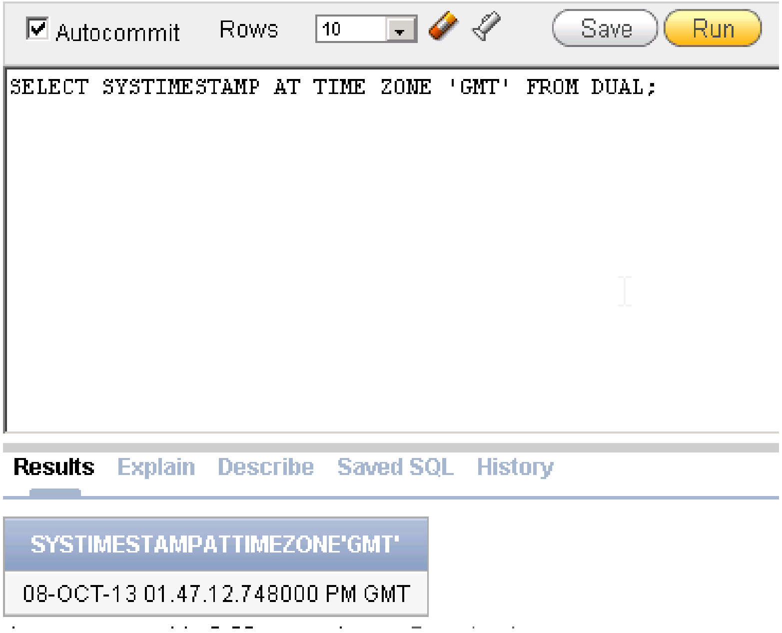
```
SELECT Order_Id, Order_Total,
TRUNC(SYSDATE) - TO_DATE(Order_TimesTamp, 'mm/dd/yyyy') AS Days_Since_Order
FROM Demo_Orders;
```

Results Explain Describe Saved SQL History

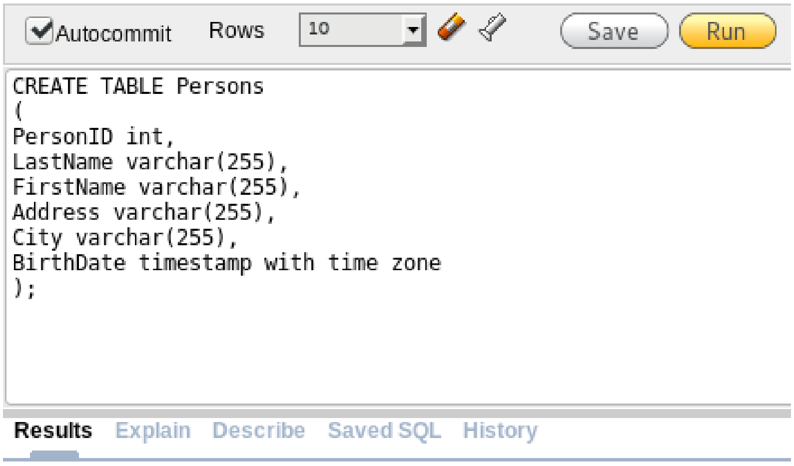
ORDER_ID	ORDER_TOTAL	DAYS_SINCE_ORDER
1	1890	11
2	2380	14
3	1640	20
4	1090	28
5	950	33
6	1515	38
7	905	48
8	1060	50
9	730	61

7.1.8.1.1 Now function based on timezone

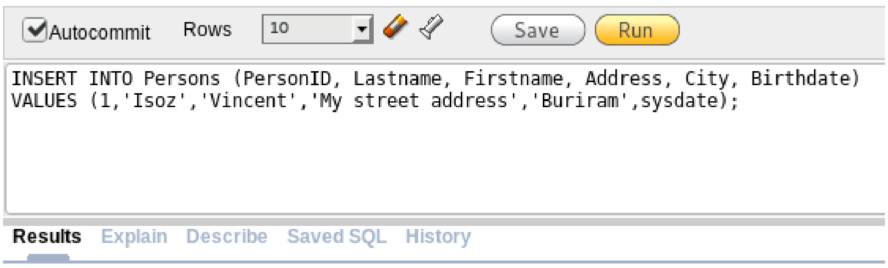
Or another typical example used a lot on the web:



It may be interesting to see how to insert such a timestamp in a database! For this purpose, let us first create a table:

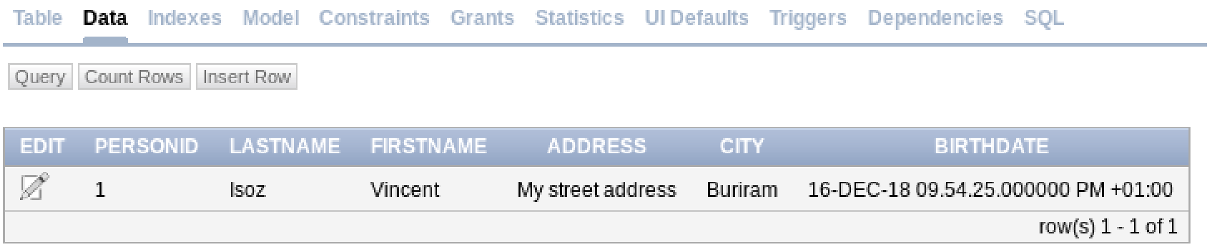


And insert a new row:



1 row(s) inserted.

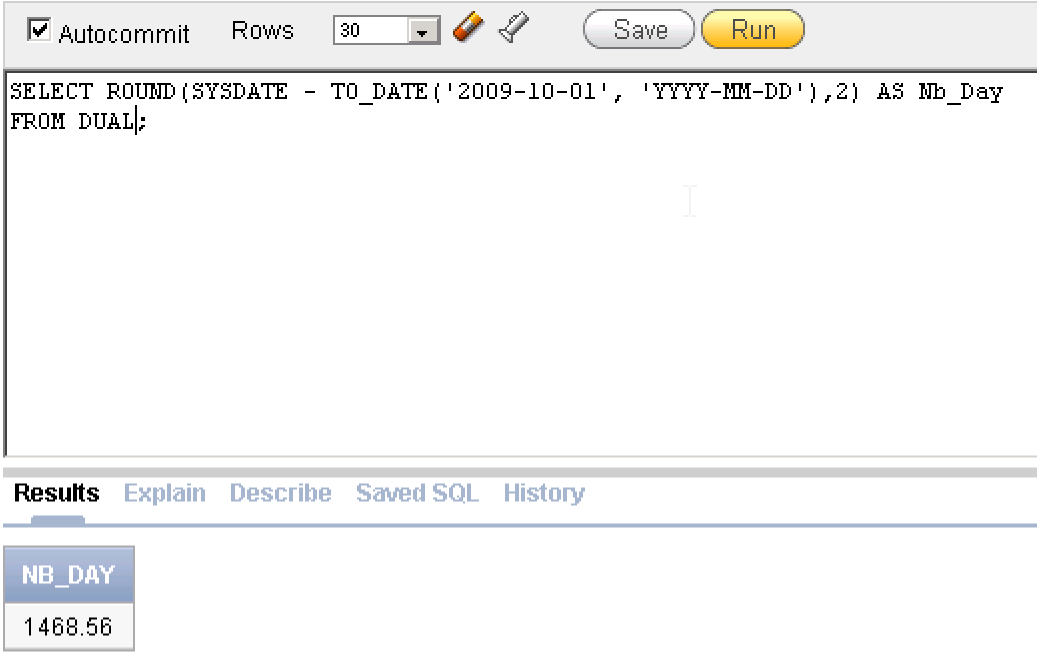
And indeed it works!



EDIT	PERSONID	LASTNAME	FIRSTNAME	ADDRESS	CITY	BIRTHDATE
	1	Isoz	Vincent	My street address	Buriram	16-DEC-18 09.54.25.000000 PM +01:00
						row(s) 1 - 1 of 1

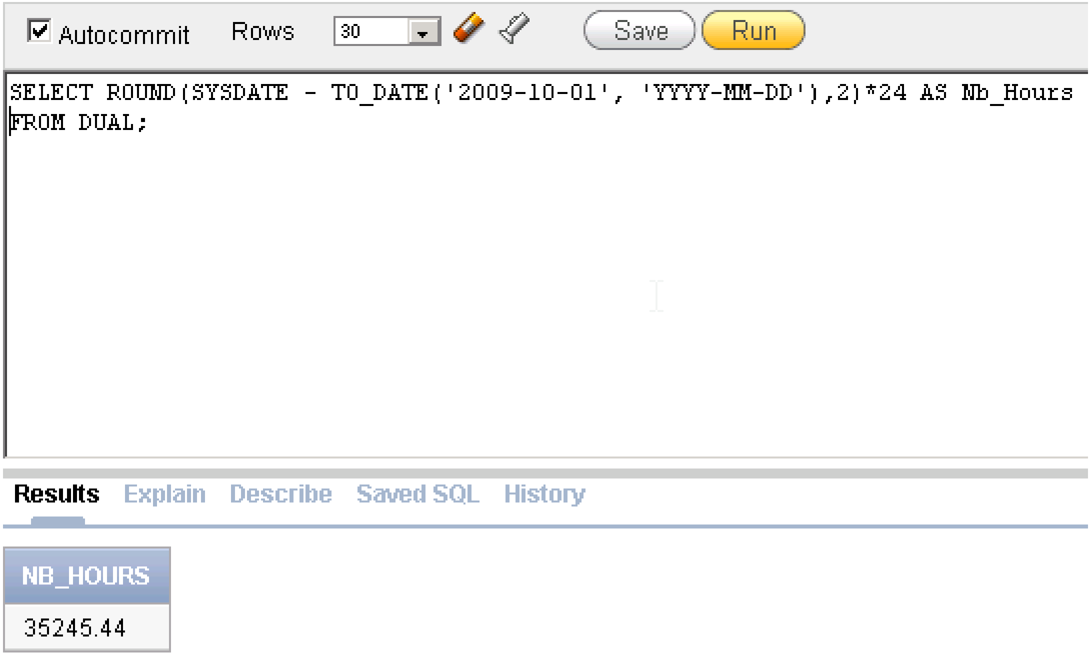
7.1.8.2 SQL Days between two dates

Here once again we will see a generic example of the function TO_DATE( ) :



7.1.8.3 SQL Hours between two dates

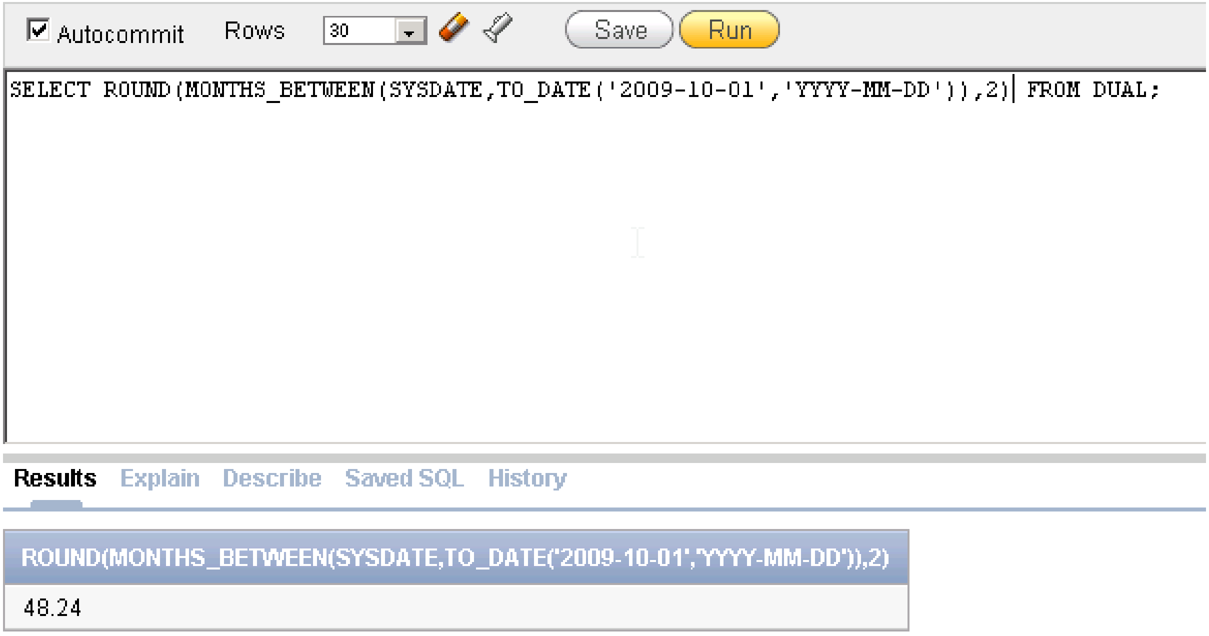
Here once again we will see a generic example:



and so on to get minutes, seconds, etc.

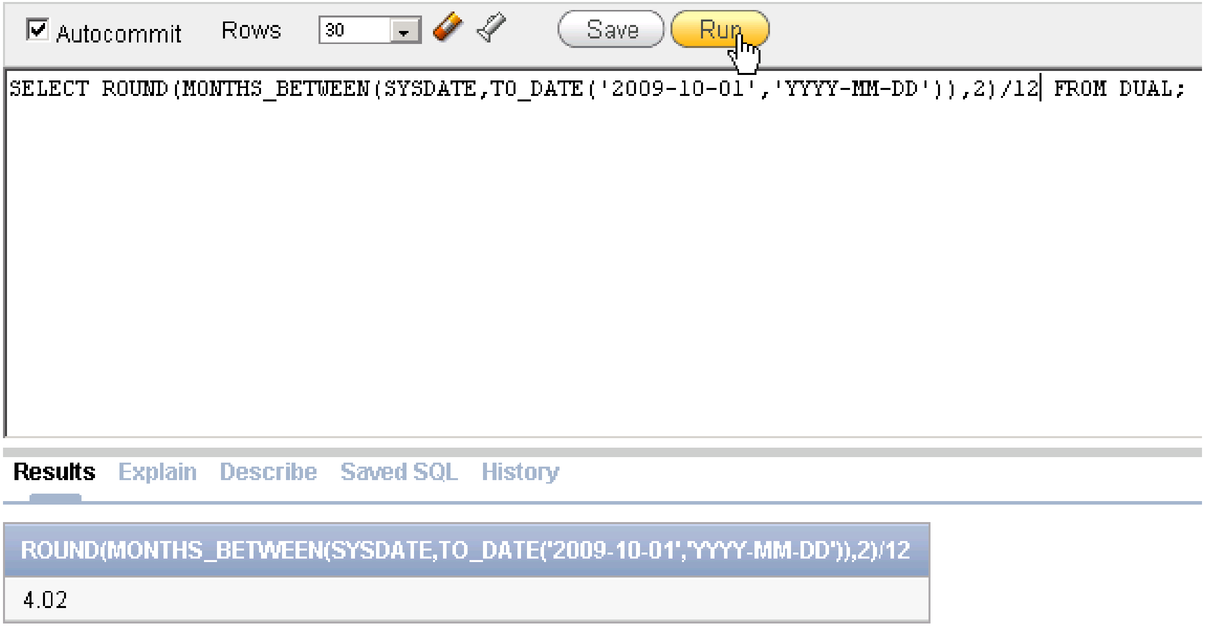
7.1.8.4 SQL Months between two dates

Here once again we will see a generic example of the function MONTHS_BETWEEN( ) :



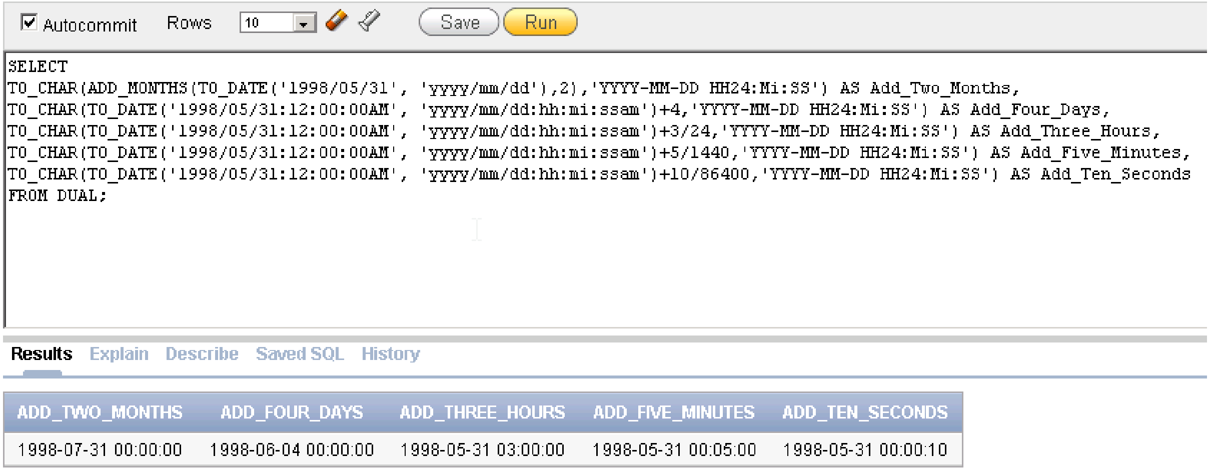
7.1.8.5 SQL Years between two dates

Here once again we will see a generic example:



7.1.8.6 SQL add a day/hour/minute/second to a date value

A generic example must be enough:




```
SELECT
TO_CHAR(ADD_MONTHS(TO_DATE('1998/05/31', 'yyyy/mm/dd'),2),'YYYY-MM-DD HH24:Mi:SS') AS Add_Two_Months,
TO_CHAR(TO_DATE('1998/05/31:12:00:00AM', 'yyyy/mm/dd:hh:mi:ssam')+4,'YYYY-MM-DD HH24:Mi:SS') AS Add_Four_Days,
TO_CHAR(TO_DATE('1998/05/31:12:00:00AM', 'yyyy/mm/dd:hh:mi:ssam')+3/24,'YYYY-MM-DD HH24:Mi:SS') AS Add_Three_Hours,
TO_CHAR(TO_DATE('1998/05/31:12:00:00AM', 'yyyy/mm/dd:hh:mi:ssam')+5/1440,'YYYY-MM-DD HH24:Mi:SS') AS Add_Five_Minutes,
TO_CHAR(TO_DATE('1998/05/31:12:00:00AM', 'yyyy/mm/dd:hh:mi:ssam')+10/86400,'YYYY-MM-DD HH24:Mi:SS') AS Add_Ten_Seconds
FROM DUAL;
```

ADD_TWO_MONTHS	ADD_FOUR_DAYS	ADD_THREE_HOURS	ADD_FIVE_MINUTES	ADD_TEN_SECONDS
1998-07-31 00:00:00	1998-06-04 00:00:00	1998-05-31 03:00:00	1998-05-31 00:05:00	1998-05-31 00:00:10

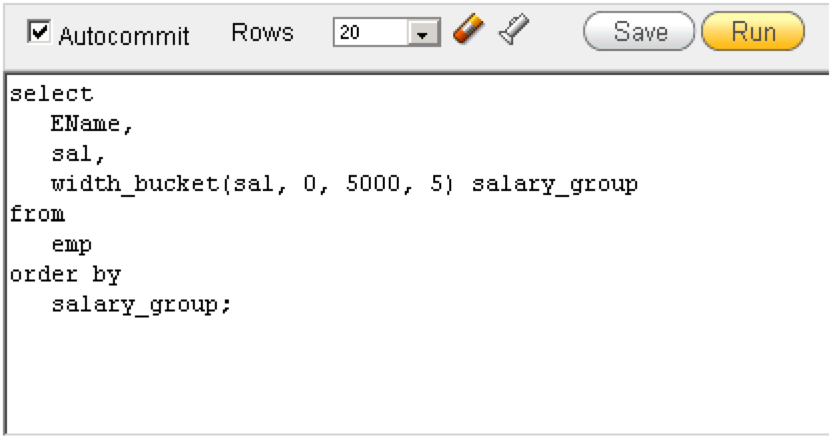
Notice that to add months, there is a specific function ADD_MONTHS( ) .

7.1.9 SQL Analytics Functions

The analytics functions of Oracle also include new statistical function of undergraduate, graduate and postgraduate level. In this chapter we will only see the functions that are not purely statistics. If the reader wants to see the related (undergraduate) statistical functions he has to go back to the chapter Statistics on page 189.

7.1.9.1 SQL WIDTH BUCKET

If you run the following query using the WIDTH_BUCKET( ) function:




```
select
   EName,
   sal,
   width_bucket(sal, 0, 5000, 5) salary_group
from
   emp
order by
   salary_group;
```

we then have the following intervals (groups):

$$\left\{ \underbrace{]-,0[}_{=0}, \underbrace{[0,1'000[}_{=1}, \underbrace{[1'000,2'000[}_{=2}, \underbrace{[2'000,3'000[}_{=3}, \underbrace{[3'000,4'000[}_{=5}, \underbrace{[4'000,5'000[}_{=6}, \underbrace{[6'000,+[}_{=7} \right\}$$

this give us for each employee the number of the group he belongs to:

Results Explain Describe Saved SQL History

ENAME	SAL	SALARY_GROUP
SMITH	800	1
ALLEN	1927.69	2
JAMES	1144.56	2
ADAMS	1100	2
TURNER	1807.21	2
MARTIN	1506.01	2
WARD	1506.01	2
MILLER	1300	2
JONES	2975	3
CLARK	2450	3
FORD	3000	4
SCOTT	3000	4
BLAKE	3433.69	4
KING	5000	6

7.1.9.2 SQL Row Number

The function ROW_NUMBER( ) gives a running serial number to a partition of records. It is very useful in reporting, especially in places where different partitions have their own serial numbers.

Here is an easy example:

Autocommit Rows 30 Save Run

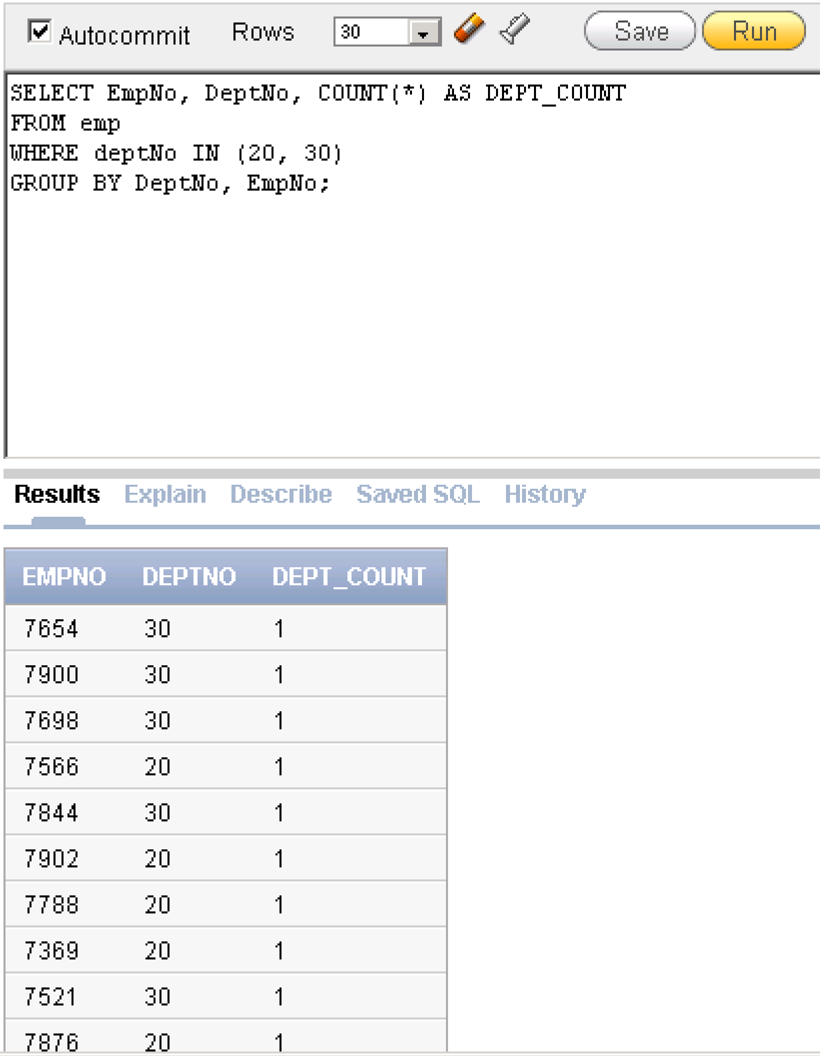
```
SELECT EmpNo, DeptNo, HireDate, ROW_NUMBER( ) OVER (PARTITION BY
DeptNo ORDER BY hiredate NULLS LAST) SRLNO
FROM Emp
WHERE DeptNo IN (10, 20)
ORDER BY DeptNo, SRLNO;
```

Results Explain Describe Saved SQL History

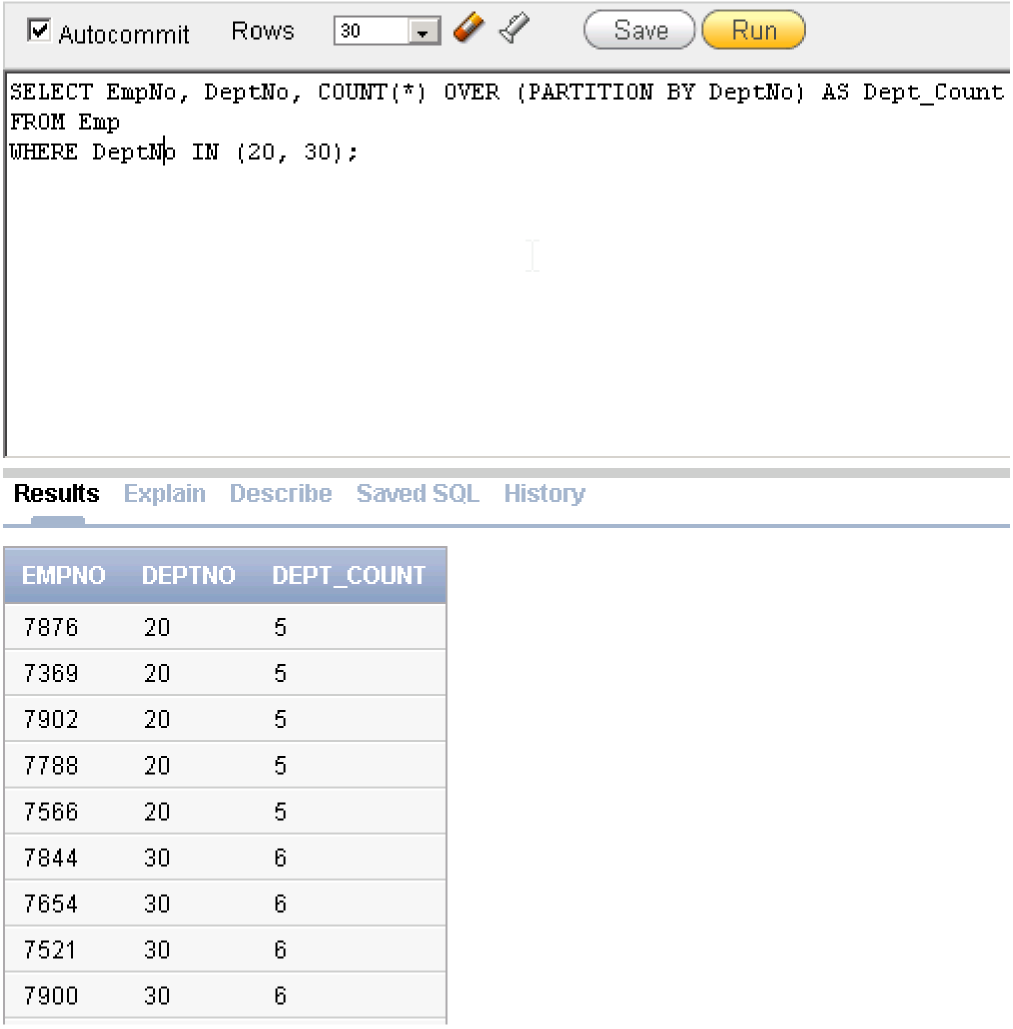
EMPNO	DEPTNO	HIREDATE	SRLNO
7782	10	06/09/1981	1
7839	10	11/17/1981	2
7934	10	01/23/1982	3
7369	20	12/17/1980	1
7566	20	04/02/1981	2
7902	20	12/03/1981	3
7788	20	12/09/1982	4
7876	20	01/12/1983	5

7.1.9.3 SQL OVER Partition

To understand clearly the OVER statement first consider the following simple query that returns departments and their employees count:



and now run the following query doing also the same but without using the GROUP BY function:

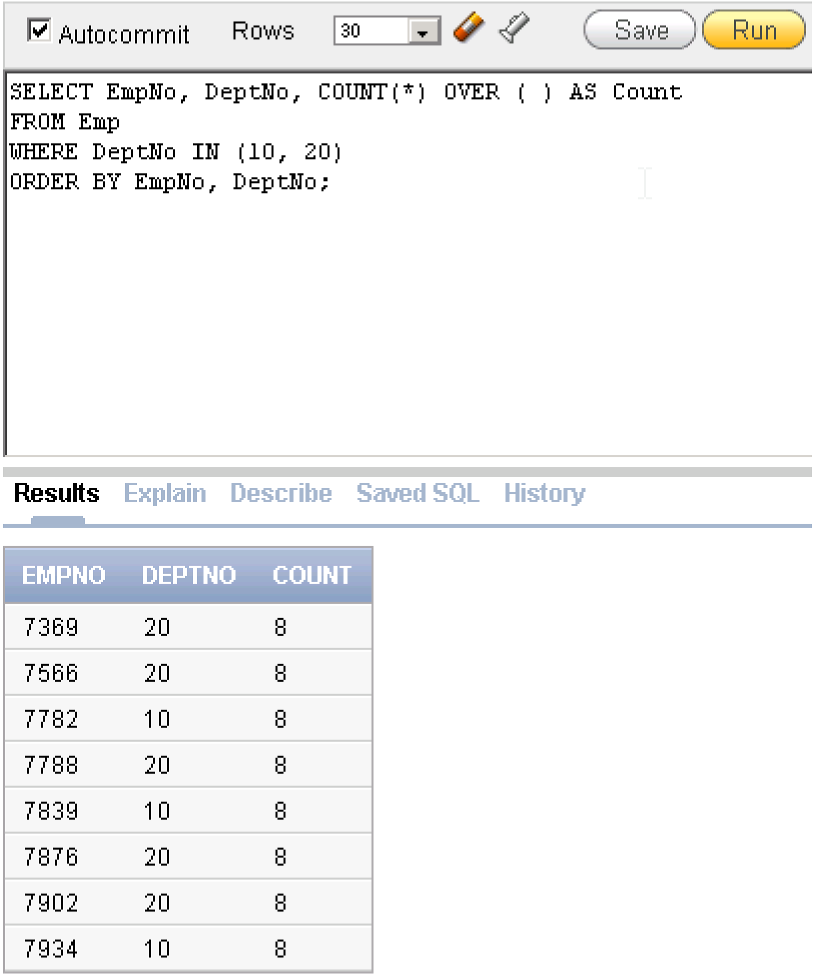



```
SELECT EmpNo, DeptNo, COUNT(*) OVER (PARTITION BY DeptNo) AS Dept_Count
FROM Emp
WHERE DeptNo IN (20, 30);
```

EMPNO	DEPTNO	DEPT_COUNT
7876	20	5
7369	20	5
7902	20	5
7788	20	5
7566	20	5
7844	30	6
7654	30	6
7521	30	6
7900	30	6

the difference should be easy to understand!

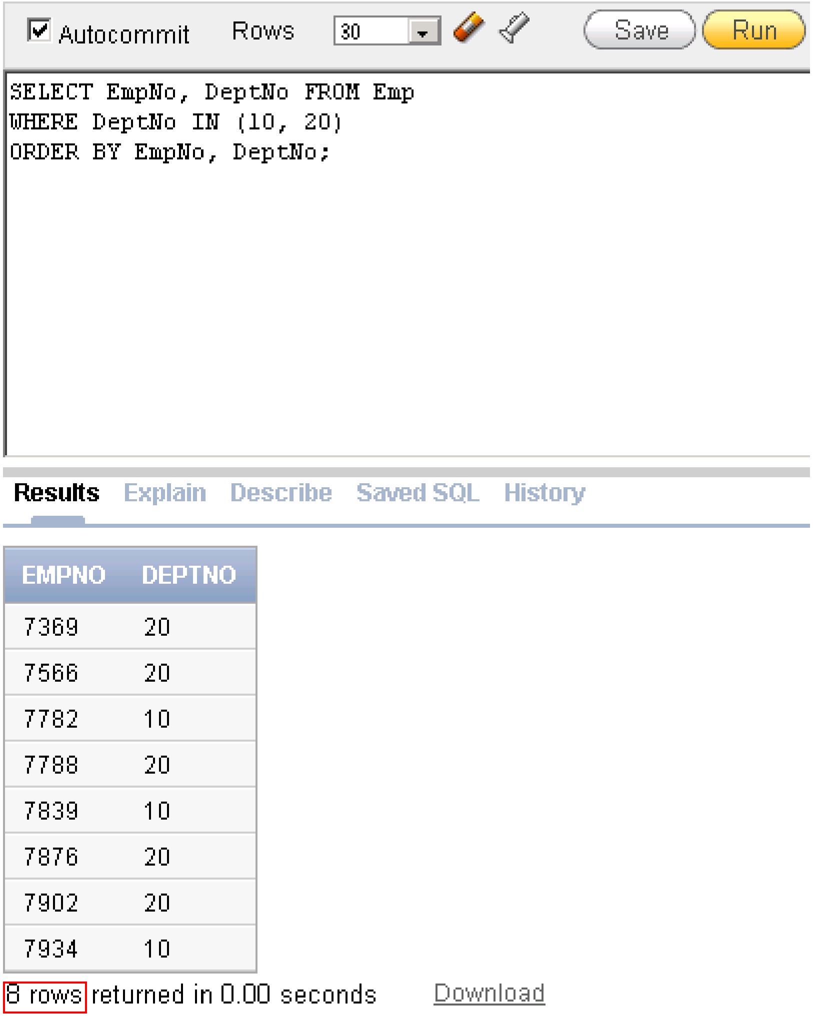
In absence of any PARTITION or <window_clause> inside the OVER( ) portion, the function acts on entire record set returned by the where clause.




```
SELECT EmpNo, DeptNo, COUNT(*) OVER ( ) AS Count
FROM Emp
WHERE DeptNo IN (10, 20)
ORDER BY EmpNo, DeptNo;
```

EMPNO	DEPTNO	COUNT
7369	20	8
7566	20	8
7782	10	8
7788	20	8
7839	10	8
7876	20	8
7902	20	8
7934	10	8

where the value "8" comes from:


Autocommit
Rows
30
Save
Run
SELECT EmpNo, DeptNo FROM Emp
WHERE DeptNo IN (10, 20)
ORDER BY EmpNo, DeptNo;
Results
Explain
Describe
Saved SQL
History
EMPNO DEPTNO
7369 20
7566 20
7782 10
7788 20
7839 10
7876 20
7902 20
7934 10
8 rows returned in 0.00 seconds
Download

7.1.9.4 SQL RANK and DENSE RANK

RANK( ) and DENSE_RANK( ) both provide rank to the records based on some column value or expression. In case of a tie of 2 records at position N, RANK( ) declares 2 positions N and skips position N+1 and gives position N+2 to the next record. While DENSE_RANK( ) declares 2 positions N but does not skip position N+1.

This is especially useful for Mann-Withney and Wilcoxon statistical rank tests! (see graduate training)

Here is an easy example to understand:


```
SELECT EmpNo, DeptNo, Sal,
ROW_NUMBER( ) OVER (PARTITION BY DeptNo ORDER BY Sal DESC NULLS LAST) AS SrlNo,
RANK() OVER (PARTITION BY DeptNo ORDER BY Sal DESC NULLS LAST) AS Rank,
DENSE_RANK() OVER (PARTITION BY DeptNo ORDER BY Sal DESC NULLS LAST) DENSE_RANK
FROM Emp
WHERE DeptNo IN (10, 20)
ORDER BY DeptNo, Rank;
```

Results Explain Describe Saved SQL History

EMPNO	DEPTNO	SAL	SRLNO	RANK	DENSE_RANK
7839	10	5000	1	1	1
7782	10	2450	2	2	2
7934	10	1300	3	3	3
7788	20	3000	1	1	1
7902	20	3000	2	1	1
7566	20	2975	3	3	2
7876	20	1100	4	4	3
7369	20	800	5	5	4

7.1.9.5 SQL LEAD and LAG

LEAD( ) has the ability to compute an expression on the next rows (rows which are going to come after the current row) and return the value to the current row. The syntax of LAG is similar except that the offset for LAG( ) goes into the previous rows.

```
LEAD (<sql_expr>, <offset>, <default>) OVER (<analytic_clause>)
```

where:

- <sql_expr> is the expression to compute from the leading row.
- <offset> is the index of the leading row relative to the current row (positive integer with default 1)
- <default> is the value to return if the <offset> points to a row outside the partition range.

Here is an easy example to understand the idea:


```
SELECT DeptNo, EmpNo, Sal,
LEAD(Sal, 1, 0) OVER (PARTITION BY DeptNo ORDER BY Sal DESC NULLS LAST) NEXT_LOWER_SAL,
LAG(Sal, 1, 0) OVER (PARTITION BY DeptNo ORDER BY Sal DESC NULLS LAST) PREV_HIGHER_SAL
FROM Emp
WHERE DeptNo IN (10, 20)
ORDER BY DeptNo, Sal DESC;
```

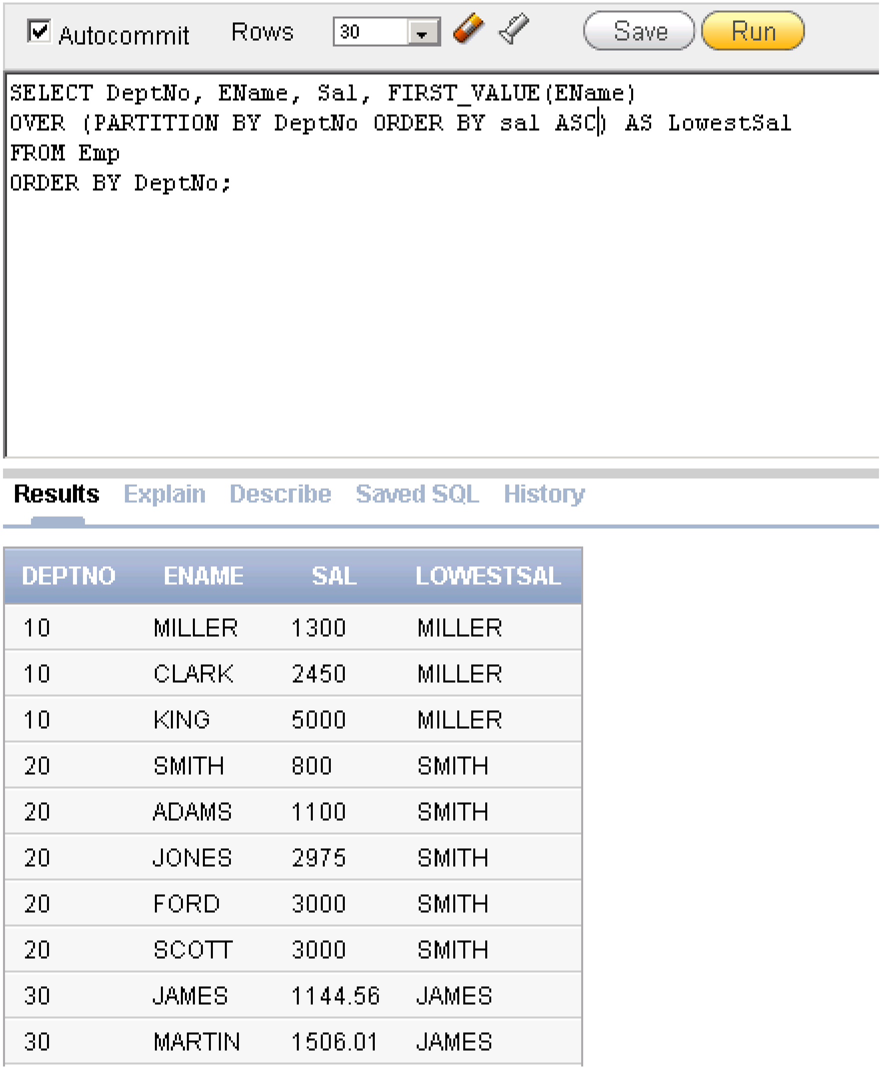
Results Explain Describe Saved SQL History

DEPTNO	EMPNO	SAL	NEXT_LOWER_SAL	PREV_HIGHER_SAL
10	7839	5000	2450	0
10	7782	2450	1300	5000
10	7934	1300	0	2450
20	7788	3000	3000	0
20	7902	3000	2975	3000
20	7566	2975	1100	3000
20	7876	1100	800	2975
20	7369	800	0	1100

7.1.9.6 SQL First Value

The function FIRST_VALUE( ) returns the first value in an ordered set of values.

The following example selects, for each employee in all departments, the name of the employee with the lowest salary:




```sql
SELECT DeptNo, EName, Sal, FIRST_VALUE(EName)
OVER (PARTITION BY DeptNo ORDER BY sal ASC) AS LowestSal
FROM Emp
ORDER BY DeptNo;
```

DEPTNO	ENAME	SAL	LOWESTSAL
10	MILLER	1300	MILLER
10	CLARK	2450	MILLER
10	KING	5000	MILLER
20	SMITH	800	SMITH
20	ADAMS	1100	SMITH
20	JONES	2975	SMITH
20	FORD	3000	SMITH
20	SCOTT	3000	SMITH
30	JAMES	1144.56	JAMES
30	MARTIN	1506.01	JAMES

or the opposite:

```
SELECT DeptNo, EName, Sal, FIRST_VALUE(EName)
OVER (PARTITION BY DeptNo ORDER BY sal DESC) AS BiggestSal
FROM Emp
ORDER BY DeptNo;
```

DEPTNO	ENAME	SAL	BIGGESTSAL
10	KING	5000	KING
10	CLARK	2450	KING
10	MILLER	1300	KING
20	FORD	3000	FORD
20	SCOTT	3000	FORD
20	JONES	2975	FORD
20	ADAMS	1100	FORD
20	SMITH	800	FORD
30	BLAKE	3433.69	BLAKE

7.1.9.6.1 First Value with Preceding

The first value with preceding is really important to be able to calculate growth in percentage of a numerical indicator. To see an example, first create the following table:

```
ALTER SESSION SET NLS_DATE_FORMAT= 'YYYYmmdd';
CREATE TABLE t (Time date,mb integer);
INSERT INTO t VALUES('20071117',103442);
INSERT INTO t VALUES('20071110',97592);
INSERT INTO t VALUES('20071103',93446);
INSERT INTO t VALUES('20071026',87648);
INSERT INTO t VALUES('20071020',84191);
INSERT INTO t VALUES('20071013',76495);
INSERT INTO t VALUES('20071006',72294);
COMMIT;
```

We will then get:

T

Table Data Indexes Model Constraints Grants Statistics UI Defaults Trig

Query Count Rows Insert Row

EDIT	TIME	MB
	11/17/2007	103442
	11/10/2007	97592
	11/03/2007	93446
	10/26/2007	87648
	10/20/2007	84191
	10/13/2007	76495
	10/06/2007	72294

row(s) 1 - 7 of 7

Now we can run the following non-trivial query the get growth in percentage:

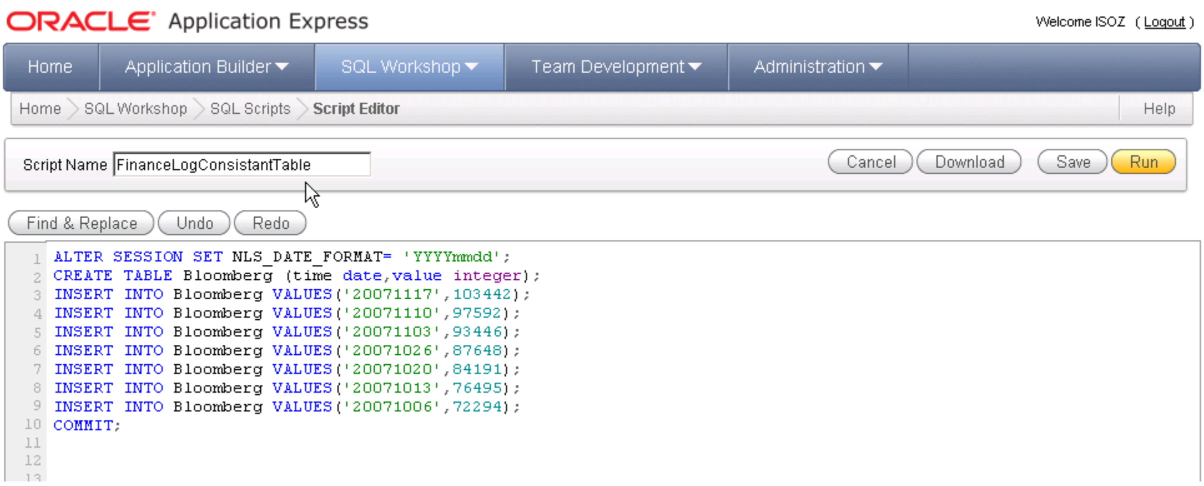
Autocommit Rows 10 Save Run

```
SELECT TIME, MAX(Mb) AS Mbytes,
ROUND((MAX(Mb)-FIRST_VALUE(MAX(Mb))
OVER(ORDER BY Time ROWS 1 PRECEDING))*100/FIRST_VALUE(MAX(Mb))
OVER(ORDER BY Time ROWS 1 PRECEDING),1) "%" FROM t
GROUP BY Time
ORDER BY Time ASC;
```

Results Explain Describe Saved SQL History

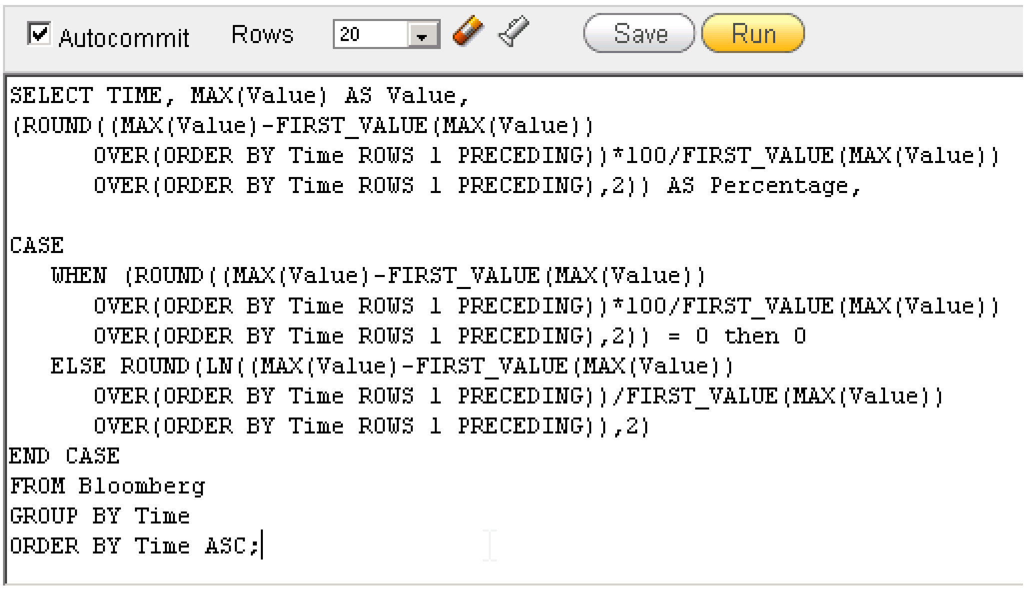
TIME	MBYTES	%
10/06/2007	72294	0
10/13/2007	76495	5.8
10/20/2007	84191	10.1
10/26/2007	87648	4.1
11/03/2007	93446	6.6
11/10/2007	97592	4.4
11/17/2007	103442	6

7.1.9.6.2 First Value with Preceding and Logarithm




```sql
ALTER SESSION SET NLS_DATE_FORMAT= 'YYYYmmdd';
CREATE TABLE Bloomberg (time date,value integer);
INSERT INTO Bloomberg VALUES('20071117',103442);
INSERT INTO Bloomberg VALUES('20071110',97592);
INSERT INTO Bloomberg VALUES('20071103',93446);
INSERT INTO Bloomberg VALUES('20071026',87648);
INSERT INTO Bloomberg VALUES('20071020',84191);
INSERT INTO Bloomberg VALUES('20071013',76495);
INSERT INTO Bloomberg VALUES('20071006',72294);
COMMIT;
```

and if we run the following non-trivial query using LN():




```sql
SELECT TIME, MAX(Value) AS Value,
(ROUND((MAX(Value)-FIRST_VALUE(MAX(Value))
      OVER(ORDER BY Time ROWS 1 PRECEDING))*100/FIRST_VALUE(MAX(Value))
      OVER(ORDER BY Time ROWS 1 PRECEDING),2)) AS Percentage,

CASE
   WHEN (ROUND((MAX(Value)-FIRST_VALUE(MAX(Value))
      OVER(ORDER BY Time ROWS 1 PRECEDING))*100/FIRST_VALUE(MAX(Value))
      OVER(ORDER BY Time ROWS 1 PRECEDING),2)) = 0 then 0
   ELSE ROUND(LN((MAX(Value)-FIRST_VALUE(MAX(Value))
      OVER(ORDER BY Time ROWS 1 PRECEDING))/FIRST_VALUE(MAX(Value))
      OVER(ORDER BY Time ROWS 1 PRECEDING)),2)
END CASE
FROM Bloomberg
GROUP BY Time
ORDER BY Time ASC;
```

we get:

TIME	VALUE	PERCENTAGE	CASE
10/06/2007	72294	0	0
10/13/2007	76495	5.81	-2.85
10/20/2007	84191	10.06	-2.3
10/26/2007	87648	4.11	-3.19
11/03/2007	93446	6.62	-2.72
11/10/2007	97592	4.44	-3.12
11/17/2007	103442	5.99	-2.81

with the famous time consistent yield used a lot in finance (we just have to change the sign but this is a detail)!

7.1.10 SQL Sytems functions (metadatas queries)

7.1.10.1 Tables size report

To obtain the list and size of all tables in MB and greater than 1 MB:

```
SELECT OWNER, TABLE_NAME, ROUND((NUM_ROWS*AVG_ROW_LEN)/(1024*1024)) MB
FROM ALL_TABLES
WHERE OWNER NOT LIKE 'SYS%'
AND NUM_ROWS> 0 AND ROUND((NUM_ROWS*AVG_ROW_LEN)/(1024*1024))>1
ORDER BY MB DESC;
```

OWNER	TABLE_NAME	MB
MDSYS	SDO_CS_SRS	5

7.1.10.2 List of columns

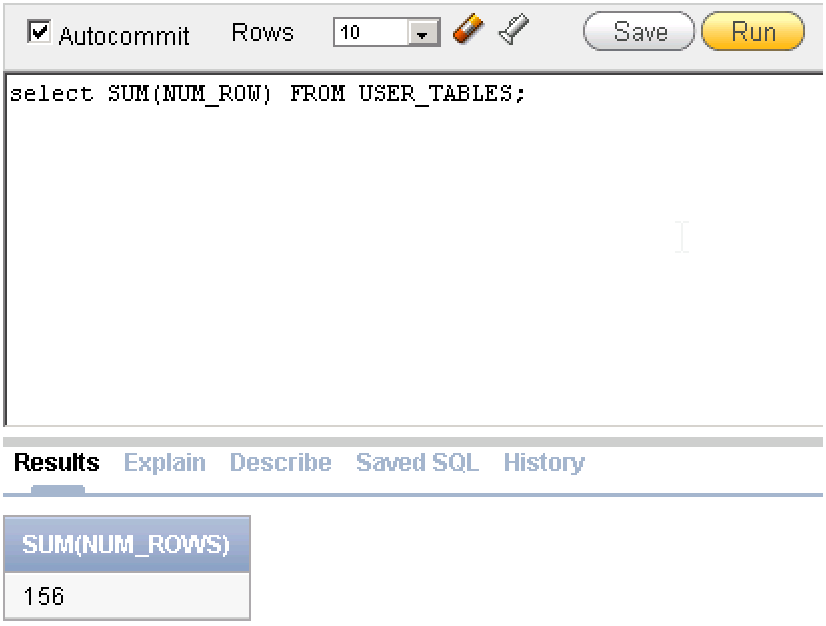
To obtain the list and size of all columns in a table:

```
SELECT COLUMN_NAME, DATA_TYPE, COLUMN_ID
FROM ALL_TAB_COLUMNS
WHERE OWNER= 'ISOZ'
AND TABLE_NAME = 'EMP';
```

COLUMN_NAME	DATA_TYPE	COLUMN_ID
EMPNO	NUMBER	1
ENAME	VARCHAR2	2
JOB	VARCHAR2	3
MGR	NUMBER	4
HIREDATE	DATE	5
SAL	NUMBER	6
COMM	NUMBER	7
DEPTNO	NUMBER	8

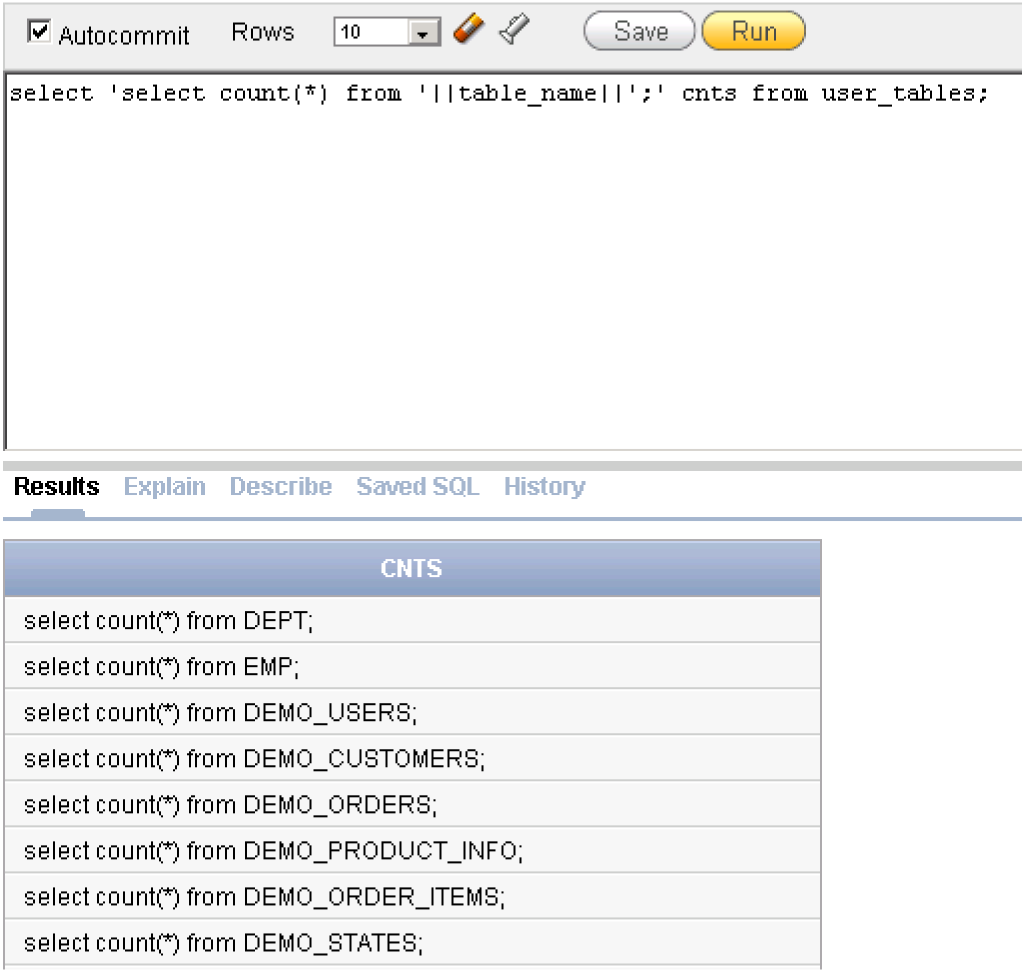
7.1.10.3 Number of rows in all tables

We already saw this query earlier above:



7.1.10.4 Generate SQL for scripting

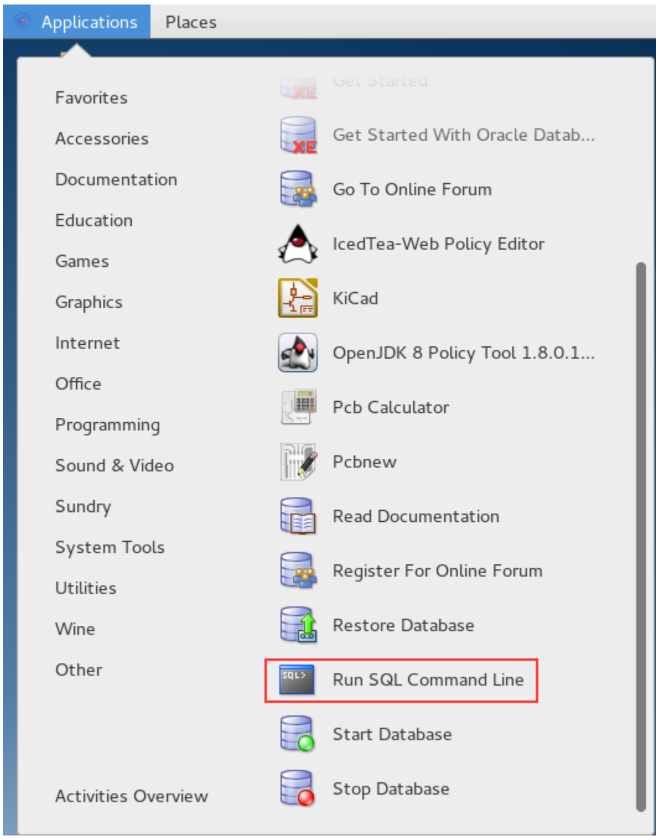
If you wish to quickly generate queries to analyze your tables by copying/pasting the code in a script you can use the following:



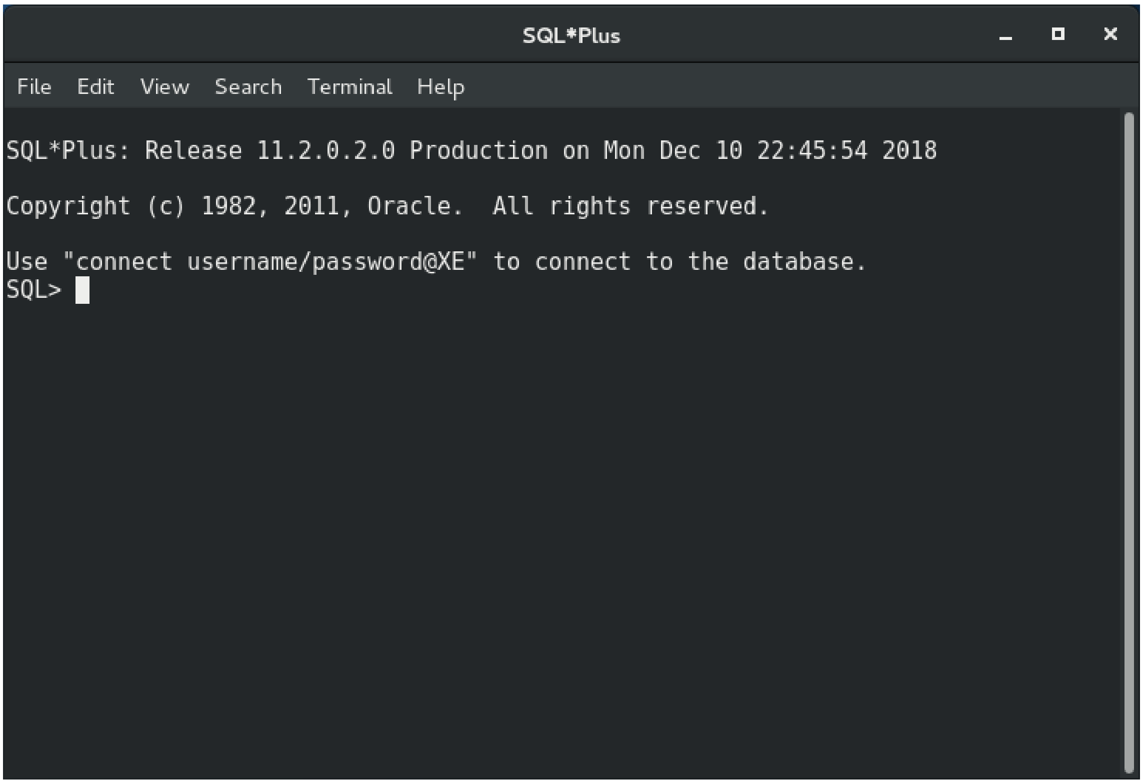
8 SQL for RDL (Rights Manipulation Language)

8.1 Create/Delete User

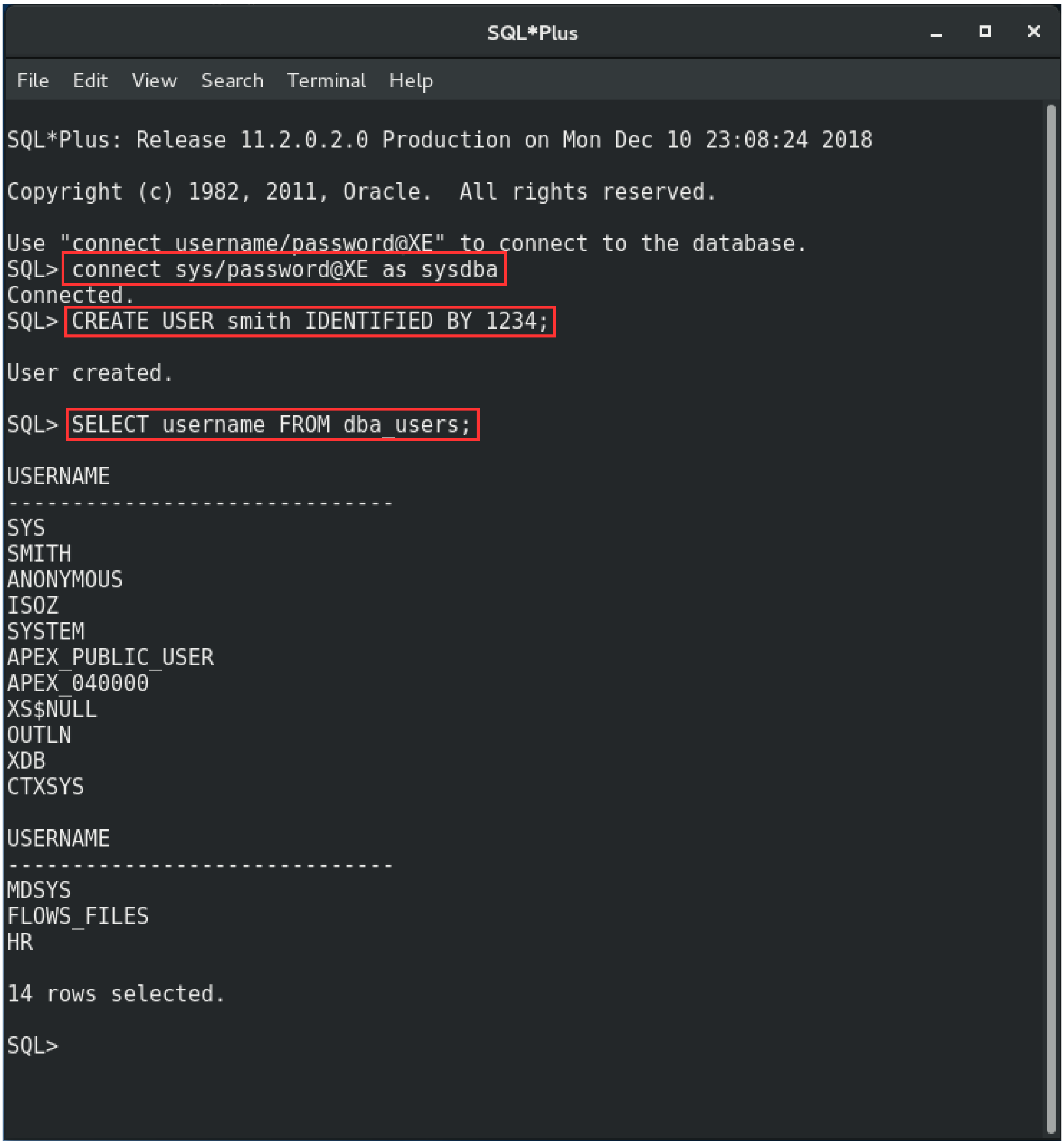
To manage users with Oracle or Oracle Express, we use SQL Plus:



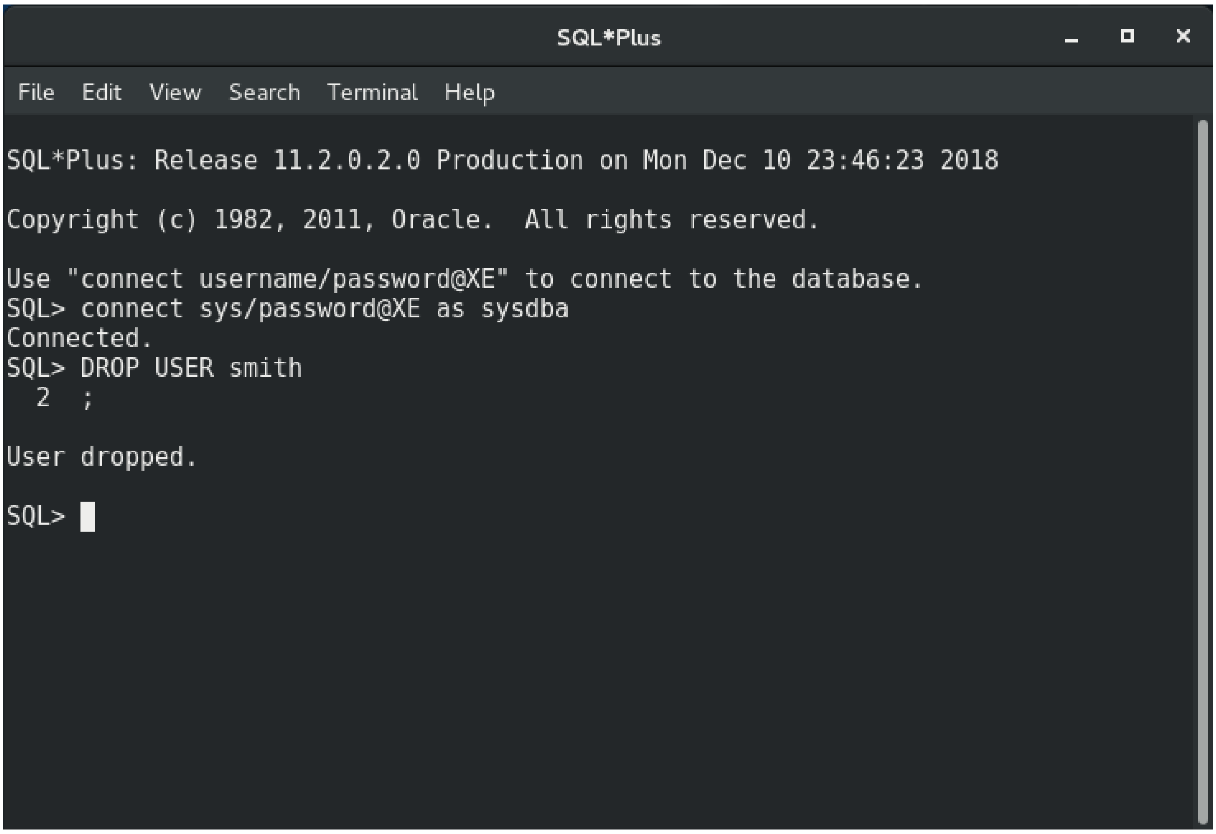
This will open:



Now we put to create a user:



Pour supprimer un utiliser on utilisera DROP USER:


SQL*Plus
File Edit View Search Terminal Help
SQL*Plus: Release 11.2.0.2.0 Production on Mon Dec 10 23:46:23 2018
Copyright (c) 1982, 2011, Oracle. All rights reserved.
Use "connect username/password@XE" to connect to the database.
SQL> connect sys/password@XE as sysdba
Connected.
SQL> DROP USER smith
2 ;
User dropped.
SQL>

8.2 Put a table in read/write

You can place a table in read-only mode with the ALTER TABLE...READ ONLY statement, and return it to read/write mode with the ALTER TABLE...READ WRITE statement. An example of a table for which read-only mode makes sense is a configuration table. If your application contains configuration tables that are not modified after installation and that must not be modified by users, your application installation scripts can place these tables in read-only mode.

The following example places the Demo_Users table in read-only mode:

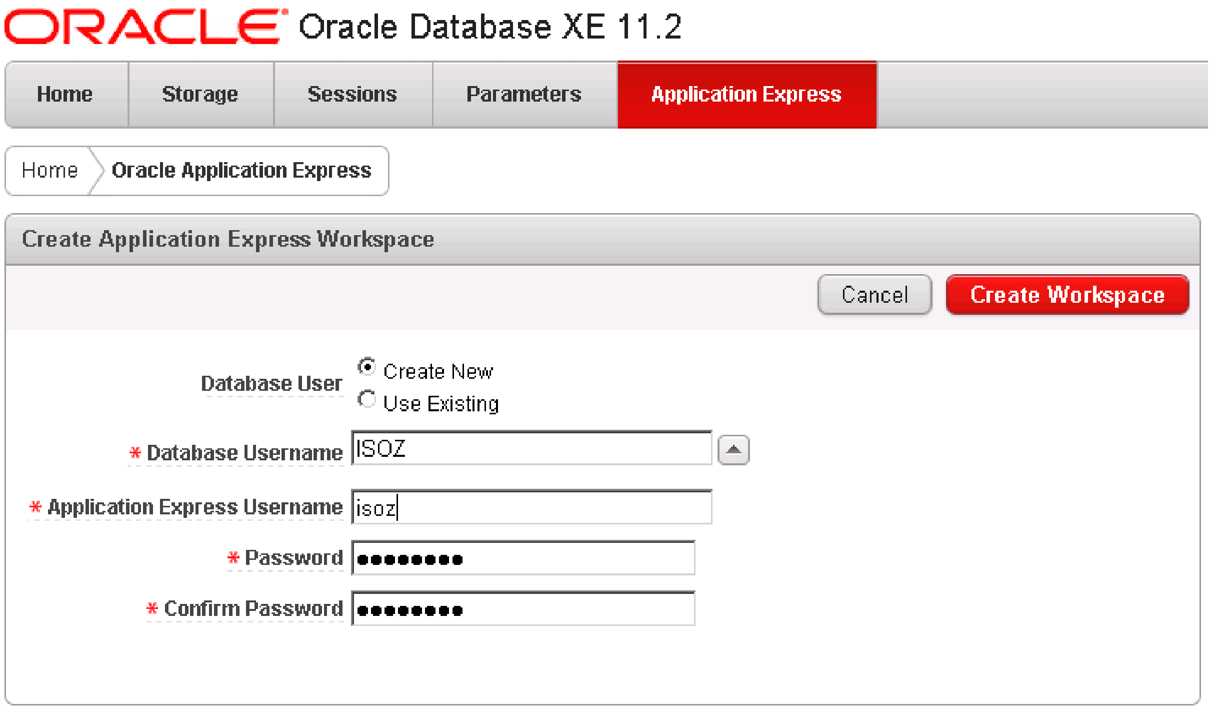
```
ALTER TABLE Demo_Users READ ONLY;
```

The following example returns the table to read/write mode:

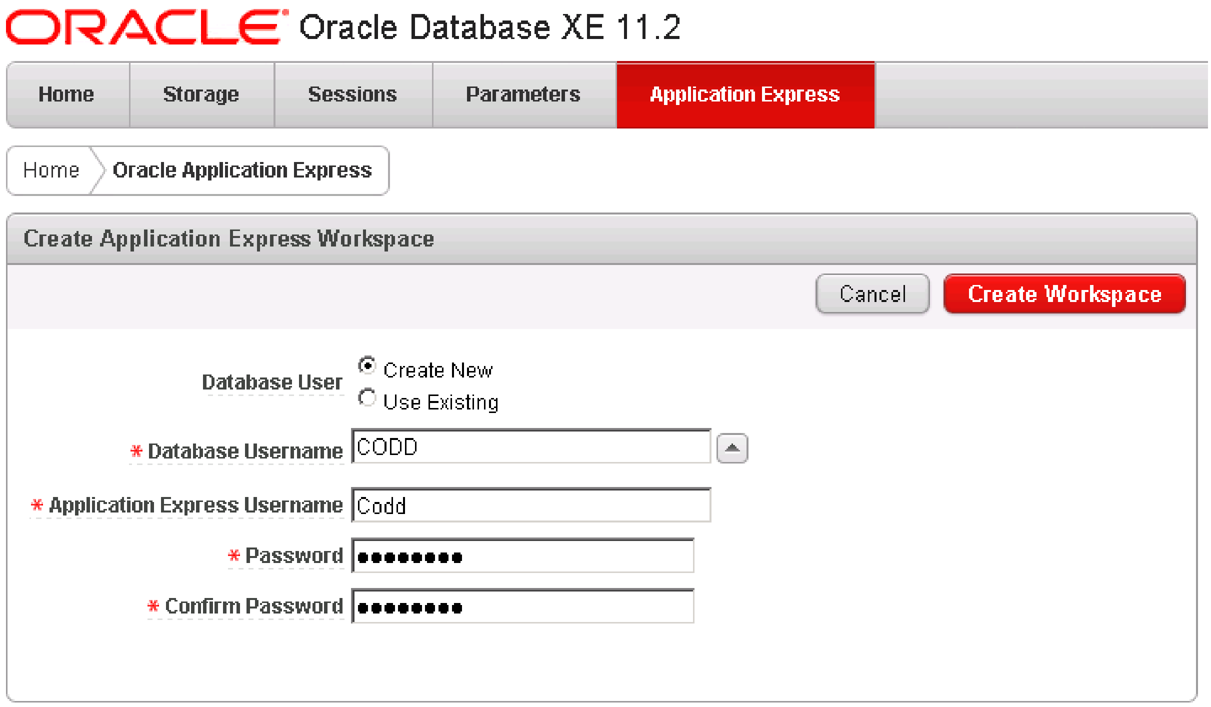
```
ALTER TABLE Demo_Users READ WRITE;
```

8.3 Grant access to tables for external users

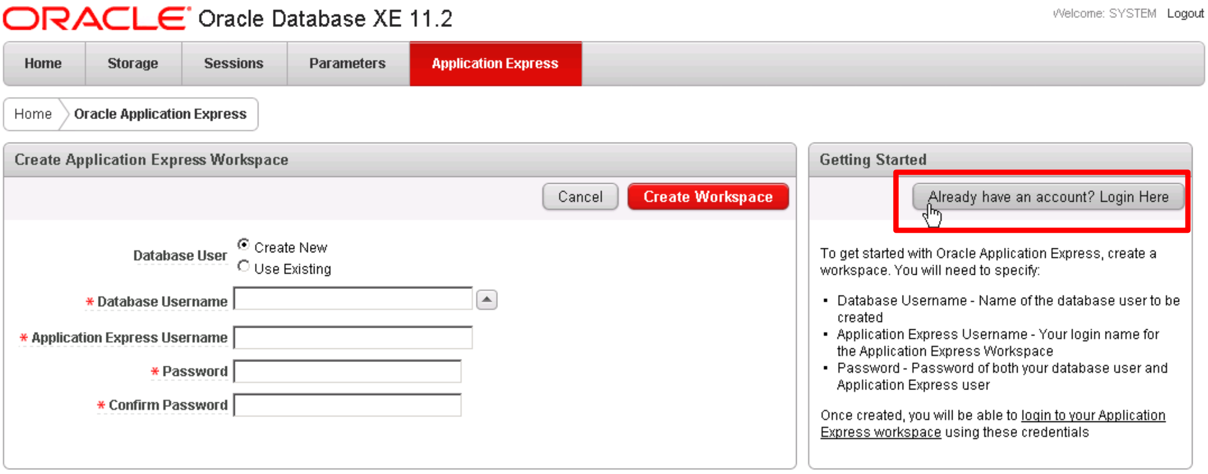
Suppose we have a database created with the following settings:



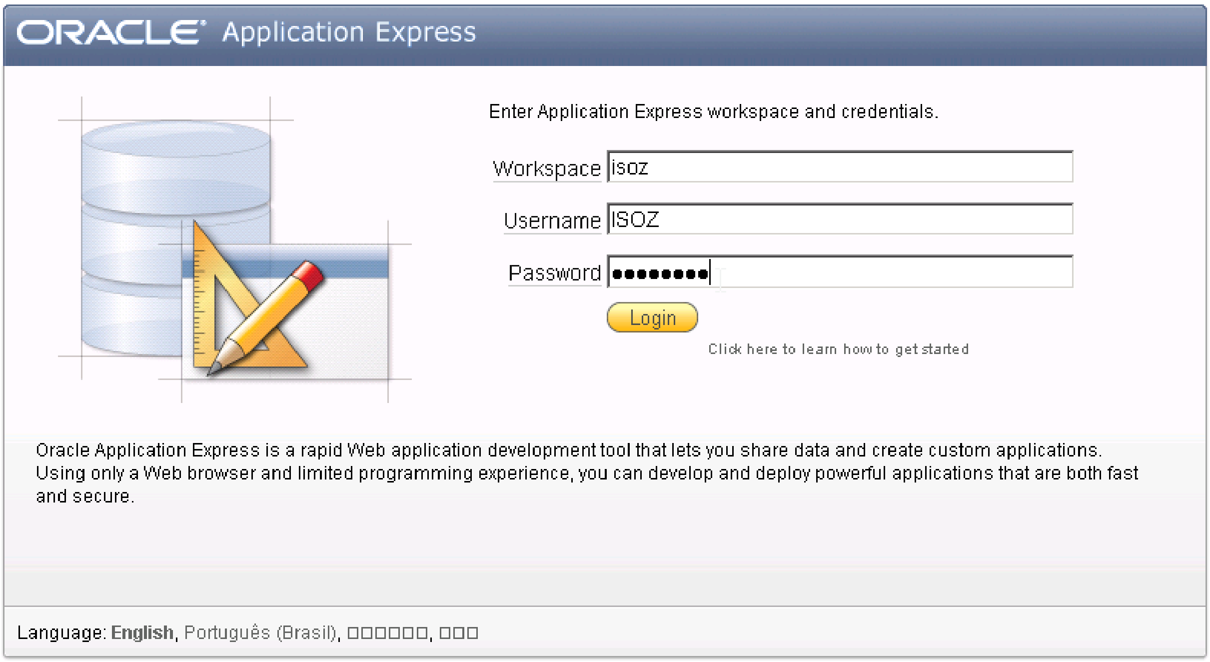
and also create the following user:



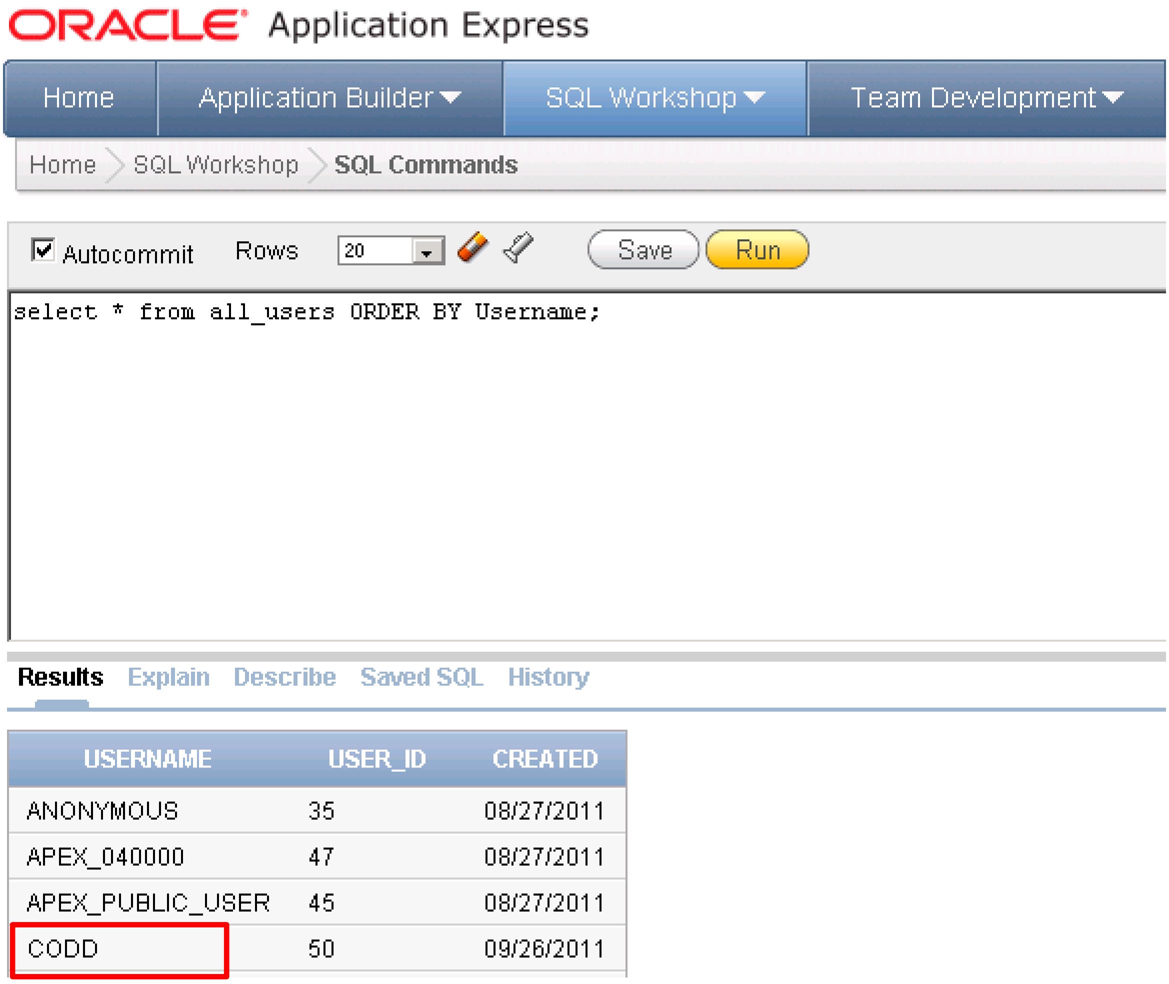
Once the user created logged into ISOZ workspace:



and type in the log fields:

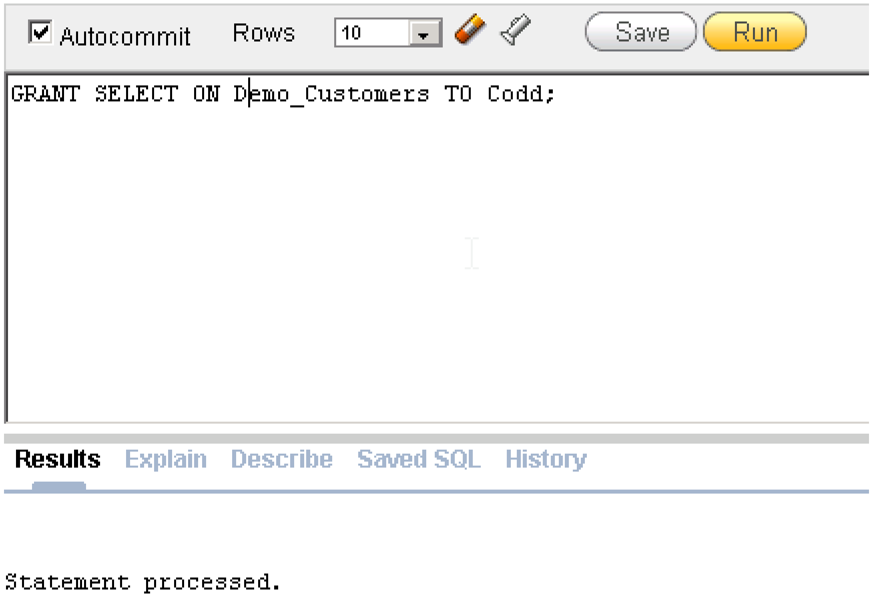


and run the following query:

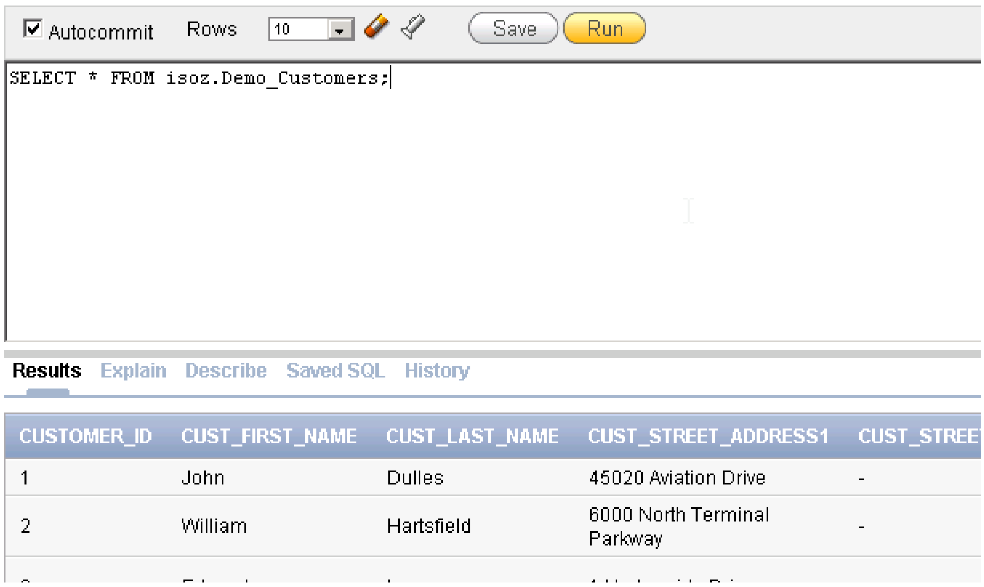


For the moment, if CODD tries to query our *Demo_Customers* table from ISOZ database he will get:

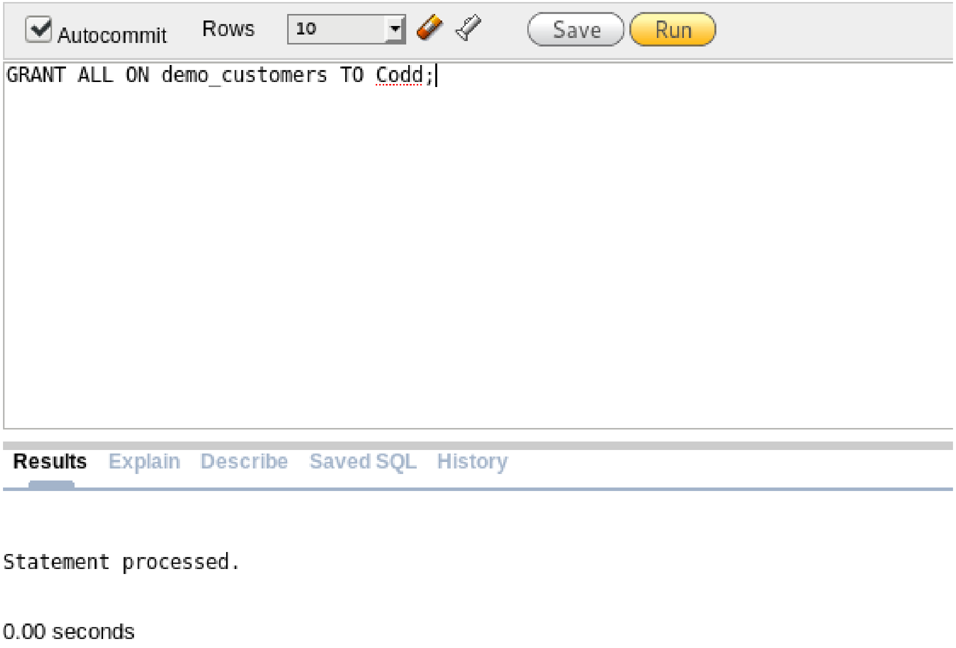
If we want to grant selection (SELECT statement) to *Codd* on our table *Demo_Customers*:



And *Codd* will be able to query *ISOZ* tables using **only** SELECT statement:



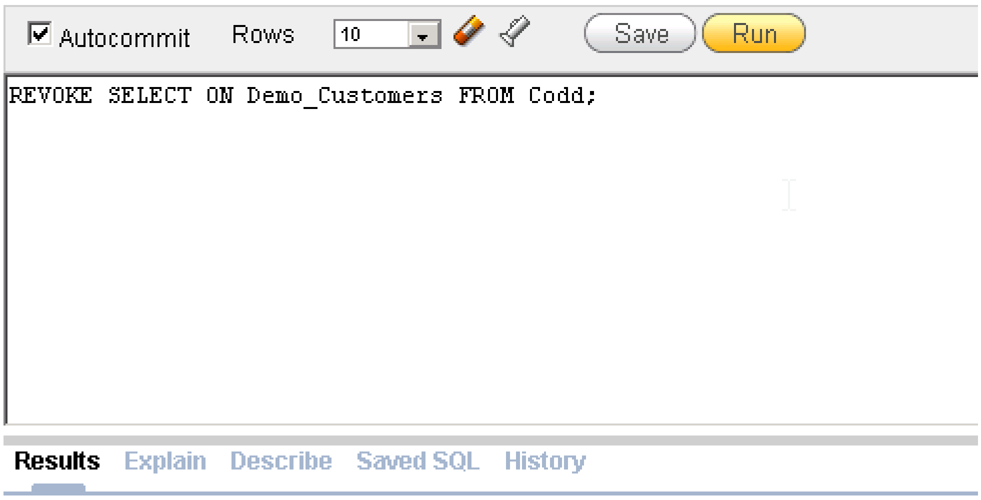
Or if you want to grant all rights:



You can then try and update SQL statement in Codd session as for example:

UPDATE isoz.Demo_Customers SET Cust_FIRST_NAME='Vincent' WHERE Cust_FIRST_NAME='Eugene';

And if we want to revoke this right:



Il y a bien une petite centaine d'accès qu'on peut donner à un utilisateur. Pour voir ces derniers concernant Oracle, le lecteur peut se référer:

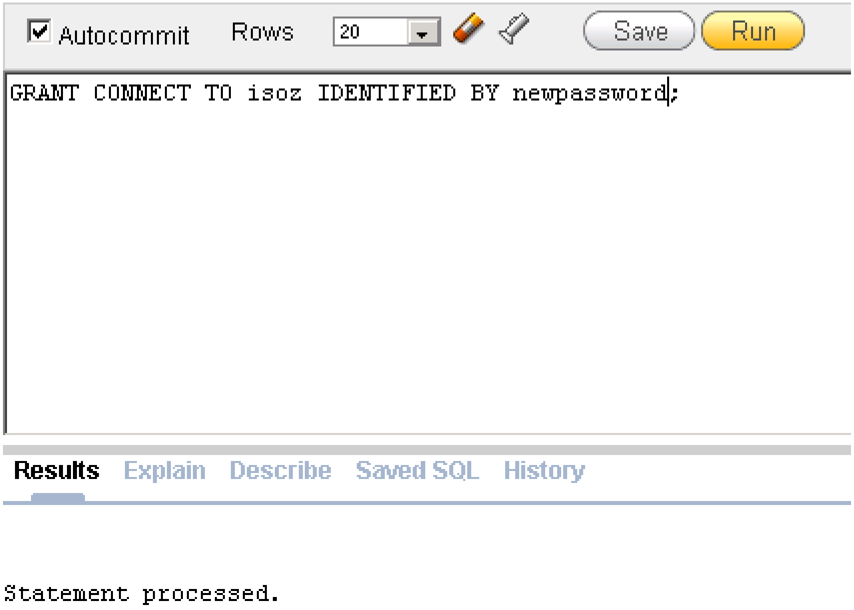
https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_9013.htm

Un petit exemple est:

```
GRANT CREATE SESSION, ALTER SESSION, CREATE DATABASE LINK, -
  CREATE MATERIALIZED VIEW, CREATE PROCEDURE, CREATE PUBLIC SYNONYM, -
  CREATE ROLE, CREATE SEQUENCE, CREATE SYNONYM, CREATE TABLE, -
  CREATE TRIGGER, CREATE TYPE, CREATE VIEW, UNLIMITED TABLESPACE -
  TO Codds;
```

8.4 Change current user password

Very simple subject! A unique example will be enough:

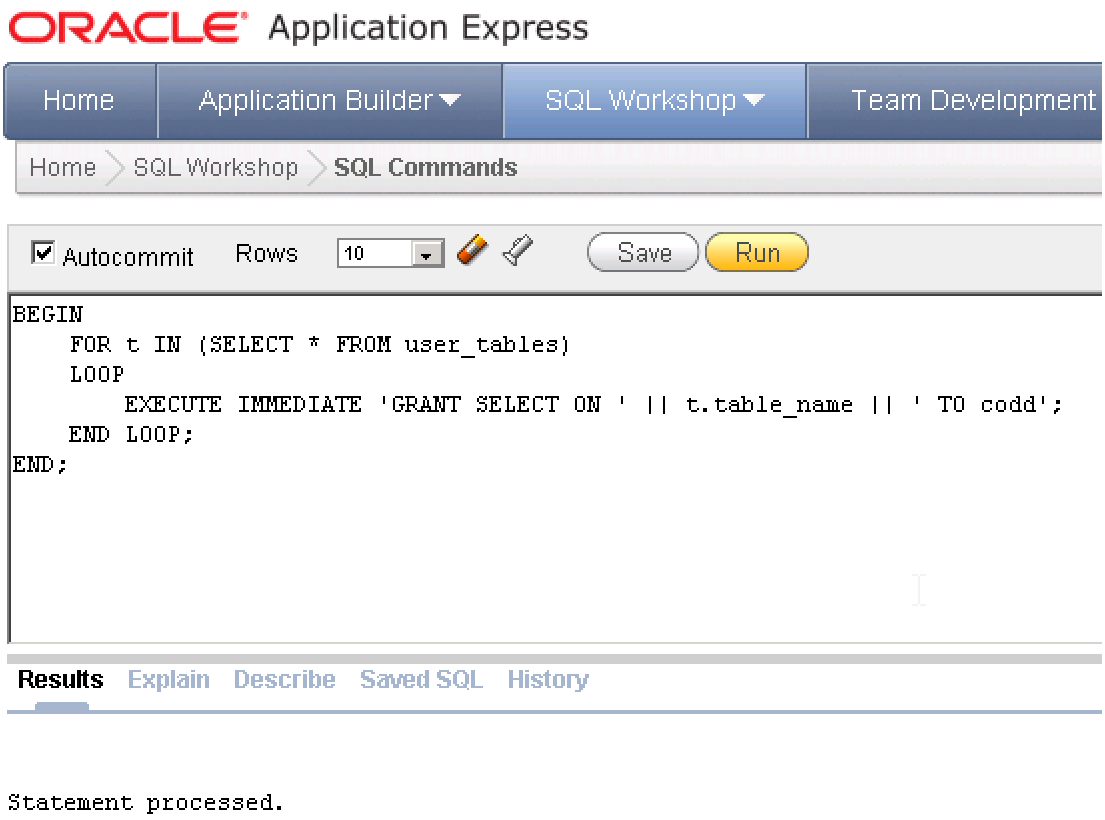


8.5 Resume of possible actions

You can grant users various privileges to tables. These privileges can be any combination of select, insert, update, delete, references, alter, and index. Below is an explanation of what each privilege means.

Privilege	Description
SELECT	Ability to query the table with a select statement.
INSERT	Ability to add new rows to the table with the insert statement.
UPDATE	Ability to update rows in the table with the update statement.
DELETE	Ability to delete rows from the table with the delete statement.
REFERENCES	Ability to create a constraint that refers to the table.
ALTER	Ability to change the table definition with the alter table statement.
INDEX	Ability to create an index on the table with the create index statement.
EXECUTE	Ability to compile the function/procedure. Ability to execute the function/procedure directly

As you can see in the example below, it is possible to create some nice procedures!:



9 PL-SQL

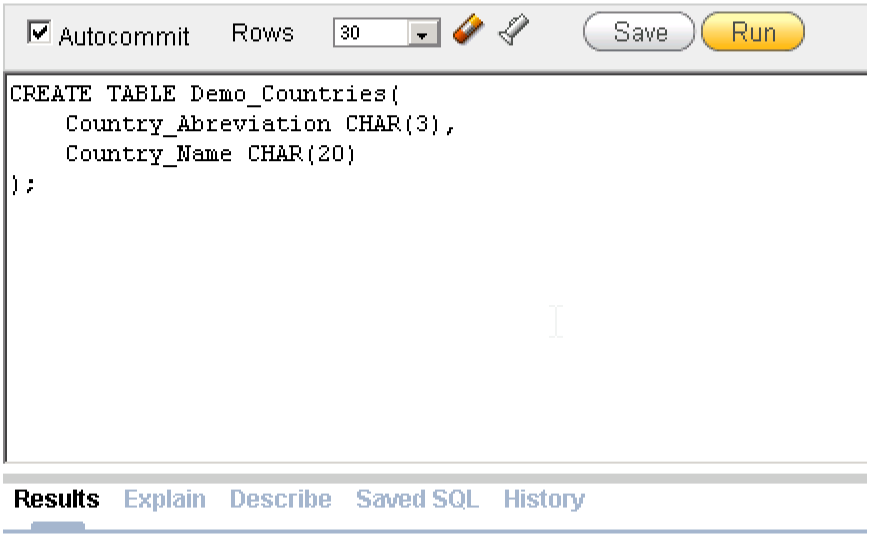
We will study PL-SQL (**Procedural Language/Structured Query Language**) in the next course... but we can still see the basics!

9.1 Create and use procedure

A procedure is a routine that an SQL user can develop to facilitate redundant operations. To see the idea a simple example could be enough.

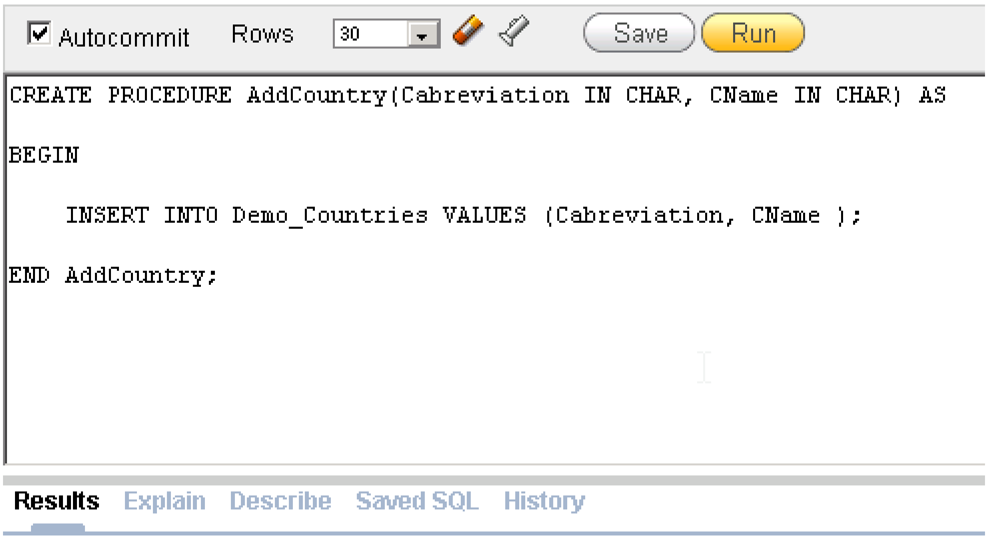
9.1.1 Procedure for data insertion (only IN variables)

First, create the small simple table as below in Oracle:



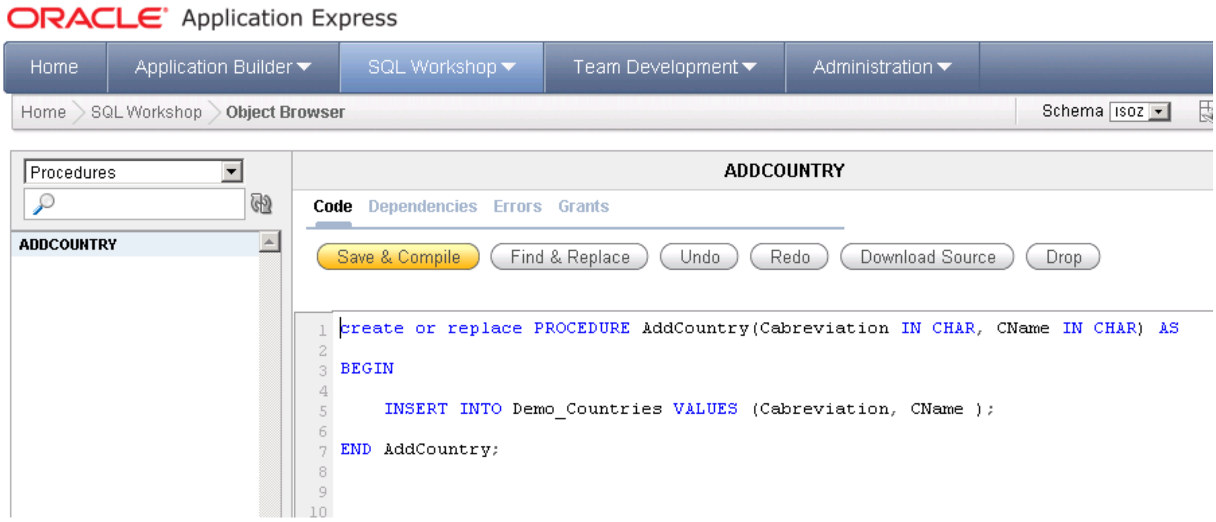

```
Table created.
```

Then create the following procedure that has for only base purpose to insert a unique row given two input values:

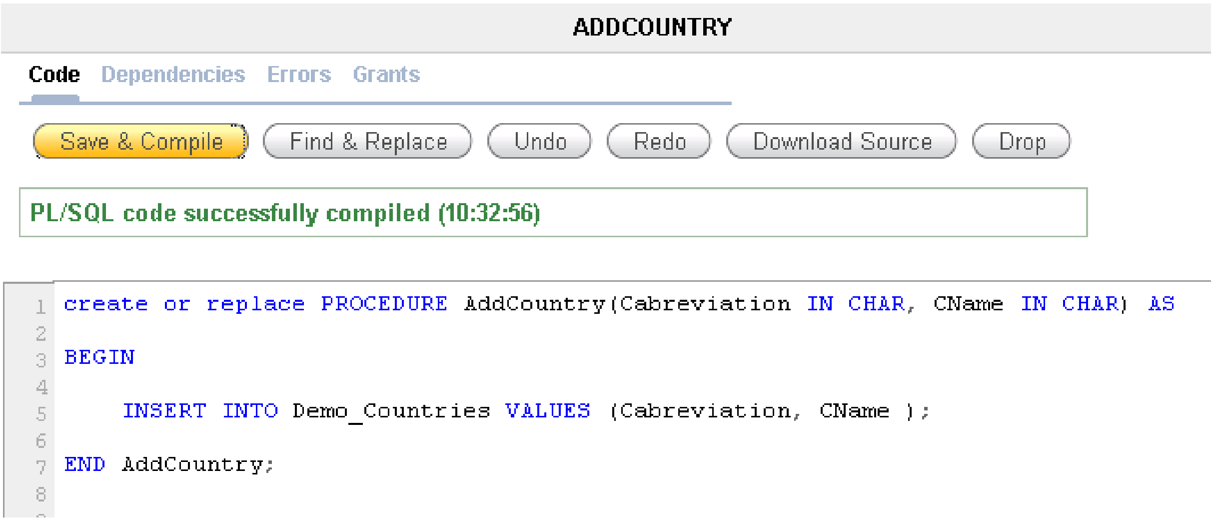



```
Procedure created.
```

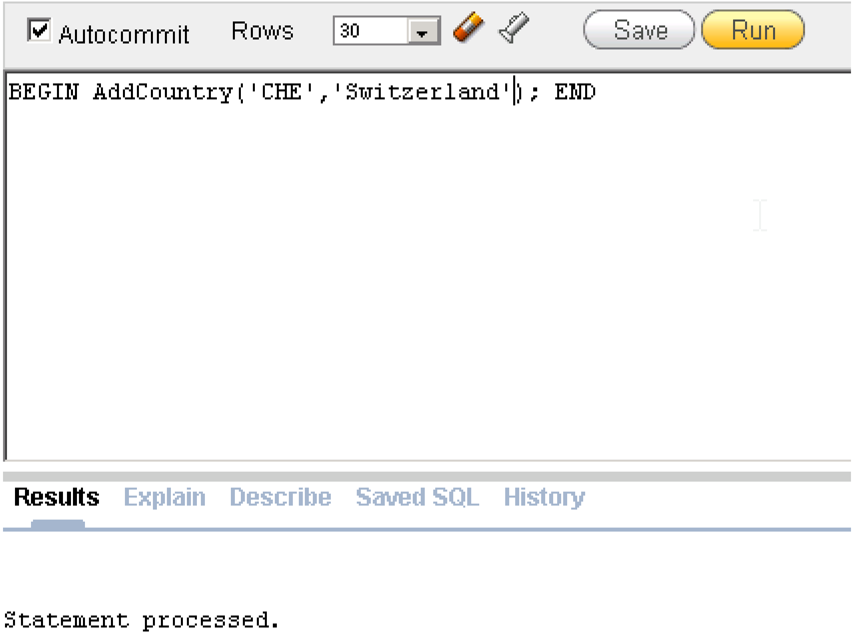
When this PL-SQL code is run you will get in the **Object Browser** of Oracle in the category **Procedures** the following:



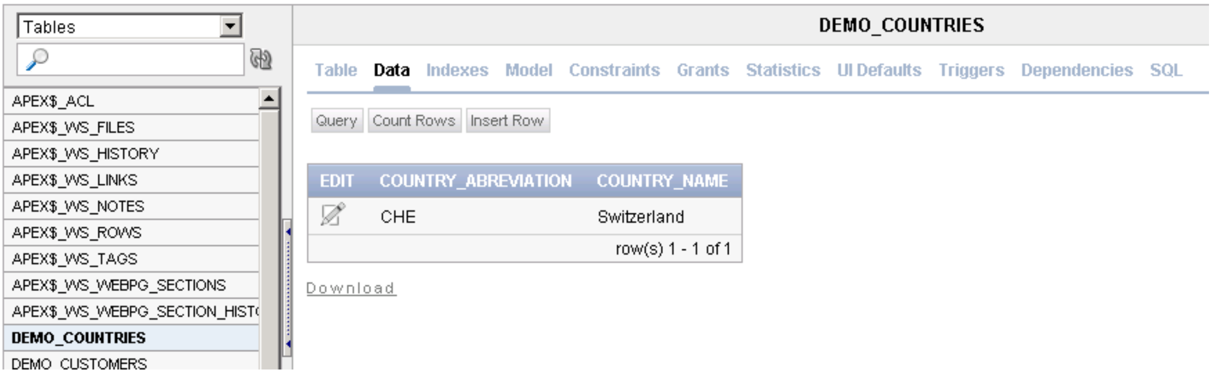
If you click on **Save & Compile** you can then check if there is an error or not before using the procedure:



Now it's time to use the procedure:



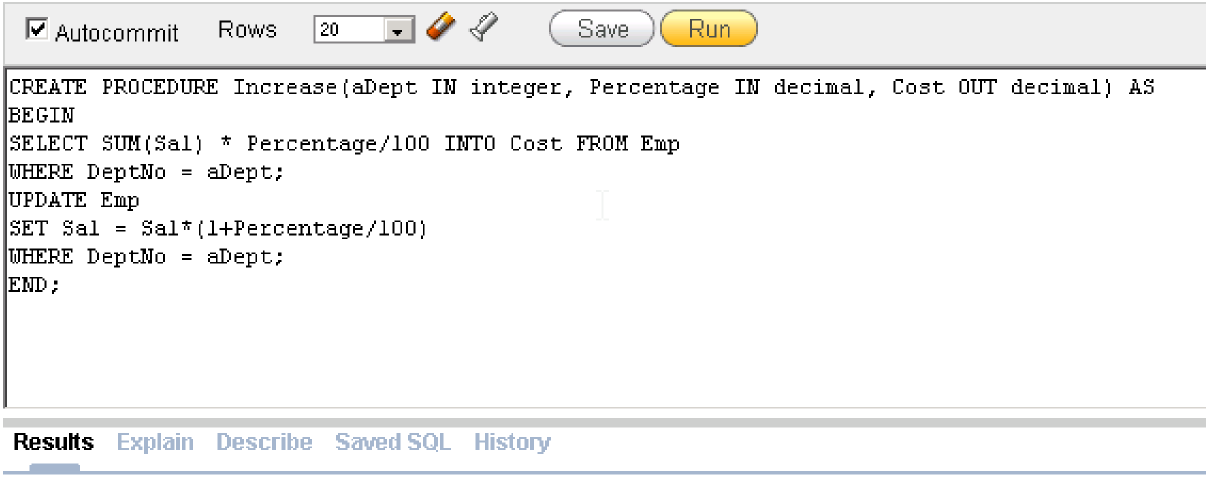
and when you run it the new line is well inserted:



and after you just play with your imagination to create what you want for procedures using all SQL statements and functions that we have study until now.

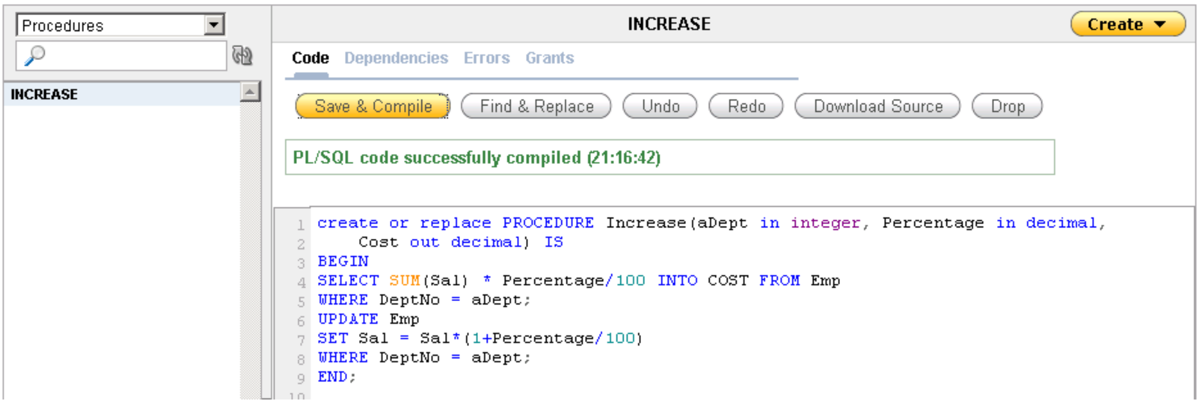
9.1.2 Procedure for data update (with IN/OUT variables)

Run the following query whose purpose is to update the salary of some given employees (who belong to a given department) and then returns the total cost for the company:

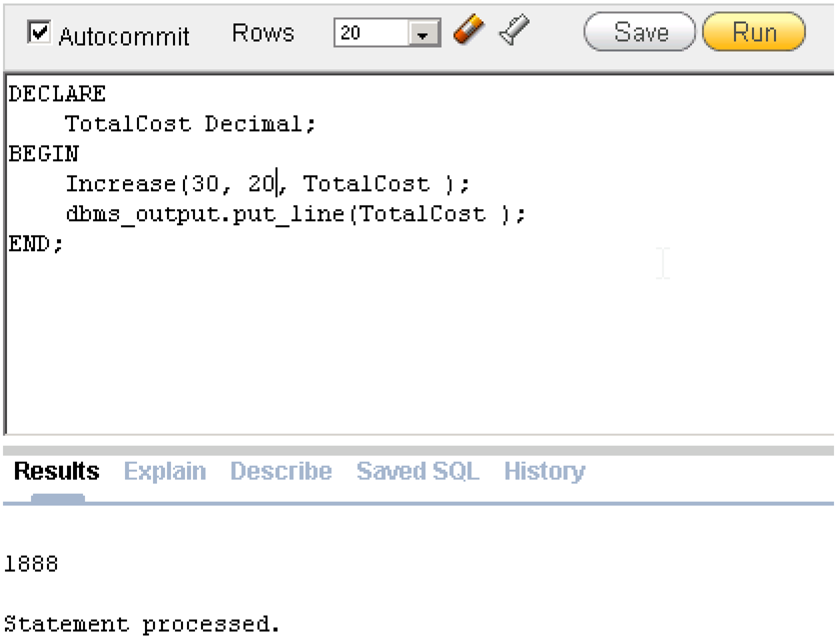



```
Procedure created.
```

Then you go in the procedures view to compile the procedure as before:

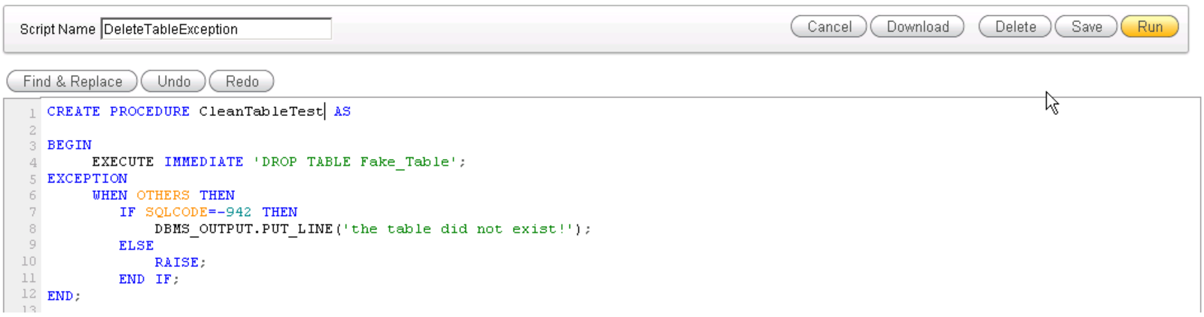


Now it's time to use the procedure:

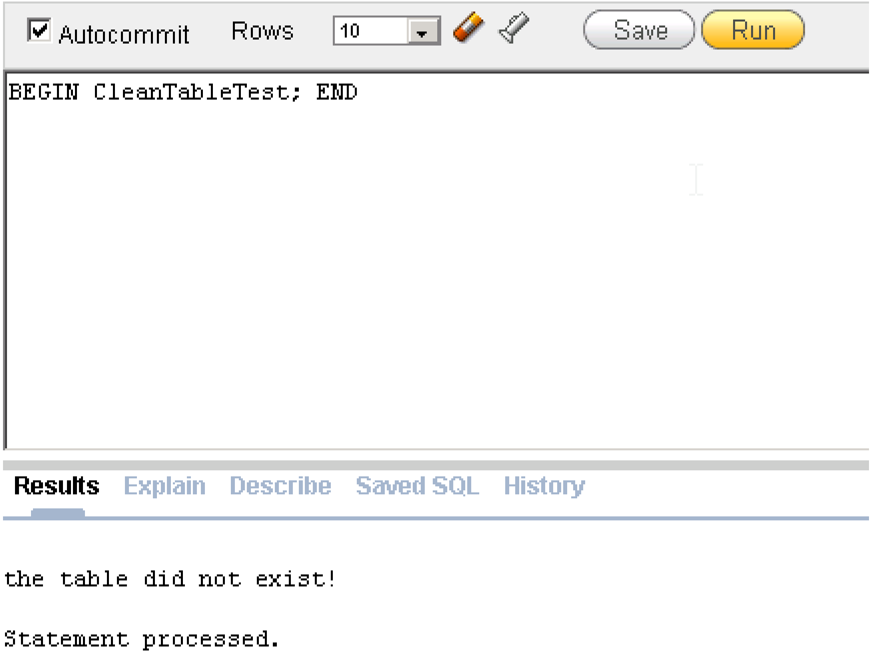


9.1.3 Procedure to check if something exists

One of my students asked me once if it is possible to check if something exists or not before acting on it? For sure! Here is a simple example with a table deletion script:



then execute the procedure:



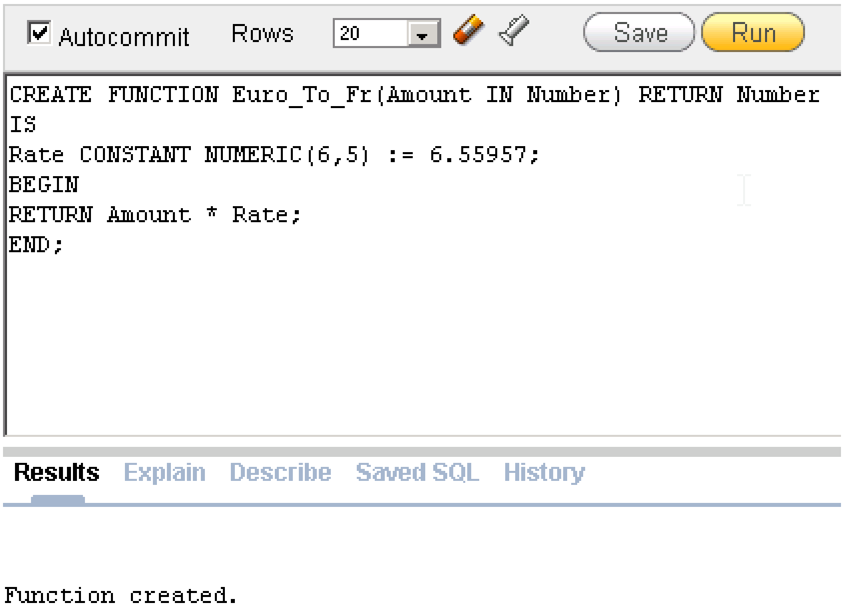
as you can see, everything works fine!

9.2 Create and use functions

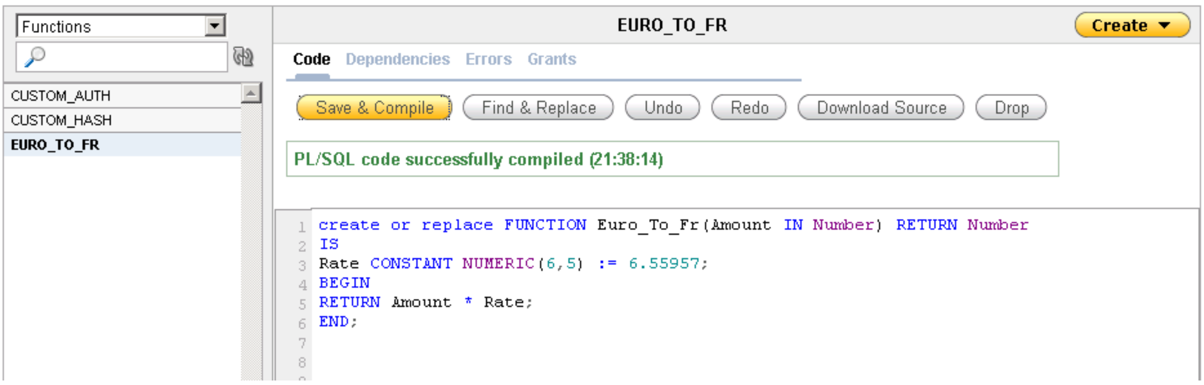
A function only makes calculation and return the value. Here is a simple example:

9.2.1 Function for data update (with IN/OUT variables)

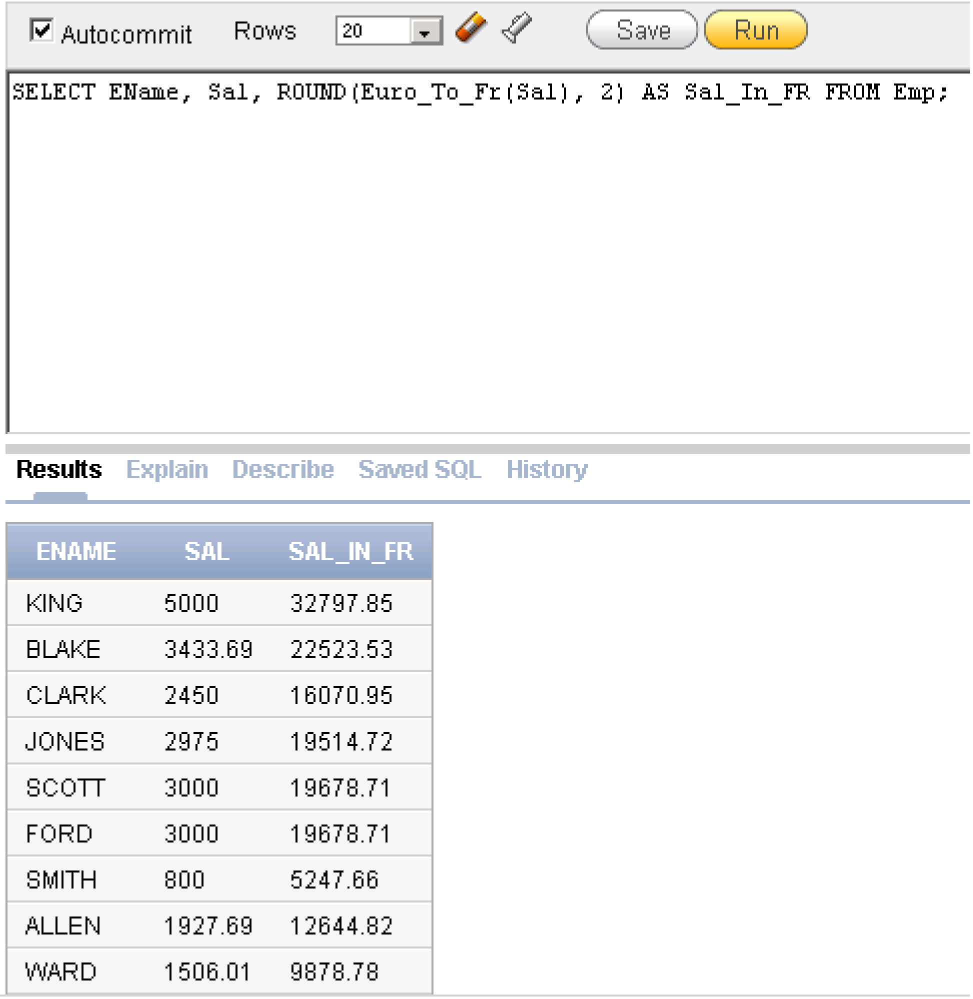
Here is a simple example that needs no explanation to be understood:



Then you get:



and if you want to use it:



That's it! Your first SQL function!

9.3 Manage Transactions

9.3.1 ACID Properties of database transaction

There are four important **properties of database transactions** these are represented by acronym **ACID** and also called **ACID properties or database transaction** where:

- **A** stands for **Atomicity**, Atom is considered to be smallest particle which cannot be broken into further pieces.database transaction has to be atomic means either all steps of transaction completes or none of them.

- **C** stands for **Consistency**, transaction must leave database in consistent state even if it succeeds or rollback.

- **I** is for **Isolation**, Two database transactions happening at same time should not affect each other and has consistent view of database. This is achieved by using isolation levels in database.

- **D** stands for **Durability**, Data has to be persisted successfully in database once transaction completed successfully and it has to be saved from power outage or other threats. This is achieved by saving data related to transaction in more than one places along with database.

9.3.1.1 *When to use database transaction with COMMIT and ROLLBACK*

Whenever any operation falls under ACID criteria you should use transactions. Many real-world scenarios require transaction mostly in banking, finance and trading domain.

Transaction control statements (TCL) manage changes made by DML statements.

What is a Transaction?

A transaction is a set of SQL statements which Oracle treats as a Single Unit. i.e. all the statements should execute successfully or none of the statements should execute.

To control transactions Oracle does not made permanent any DML statements unless you commit it. If you don't commit the transaction and power goes off or system crashes then the transaction is roll backed.

TCL Statements available in Oracle are

- **COMMIT**:Make changes done in transaction permanent.

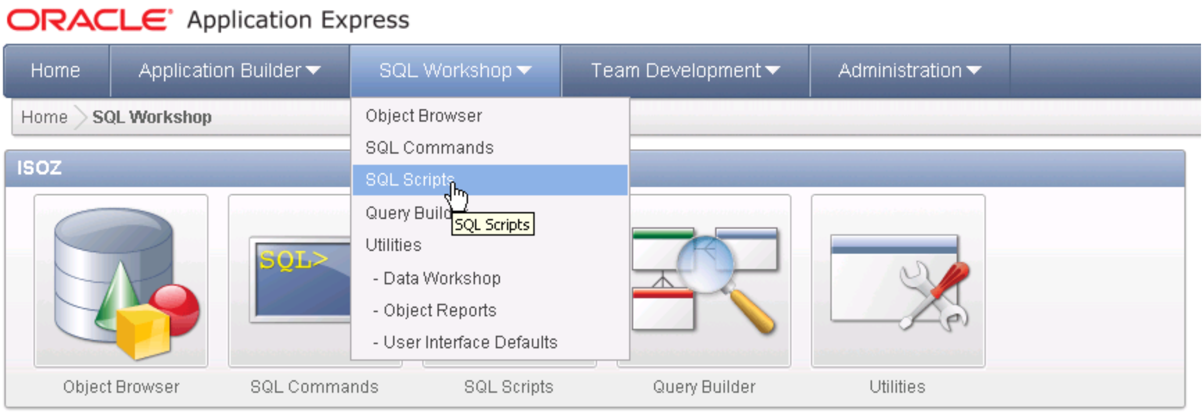
- **ROLLBACK**:Rollbacks the state of database to the last commit point.

- **SAVEPOINT**:Use to specify a point in transaction to which later you can rollback.

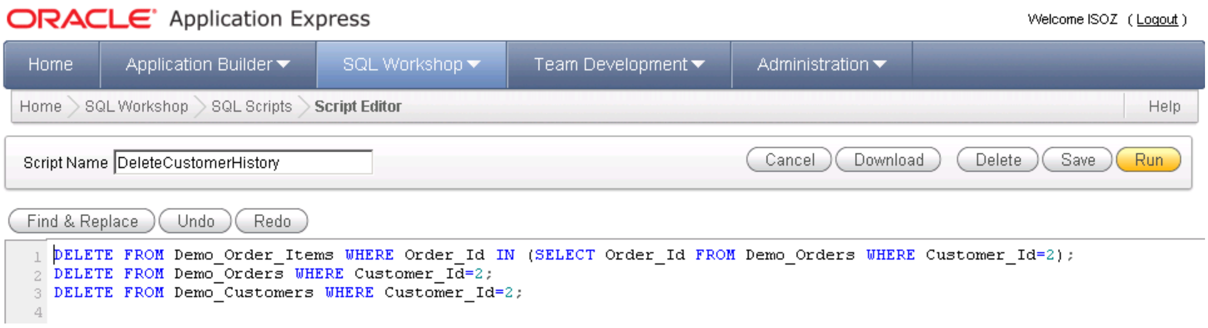
We will see here an example of such an application.

9.3.1.1.1 Simple COMMIT Example

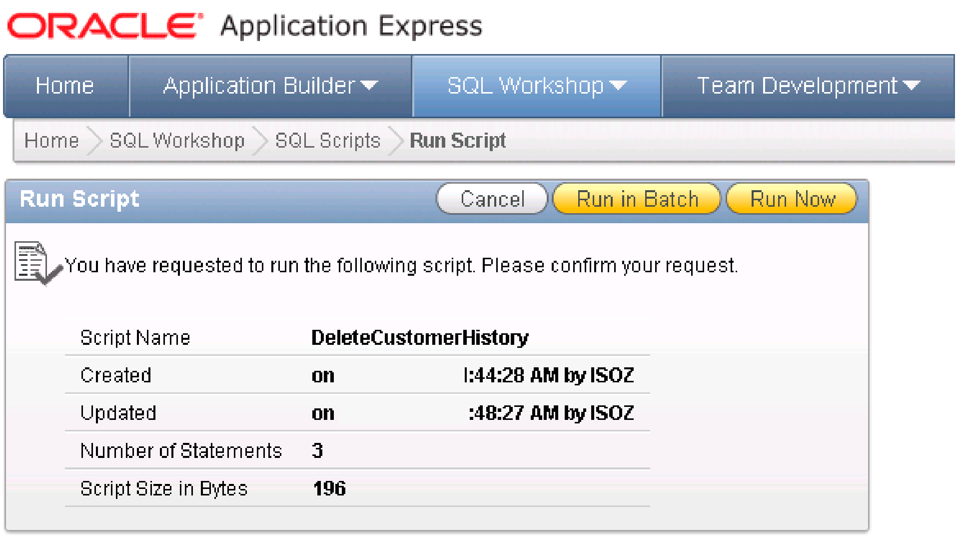
We want to delete all logs of a given customer and even the customer itself. To do this, we go first in SQL Script **to be able to run multiple queries at the same time** (what SQL Commands does not permit):



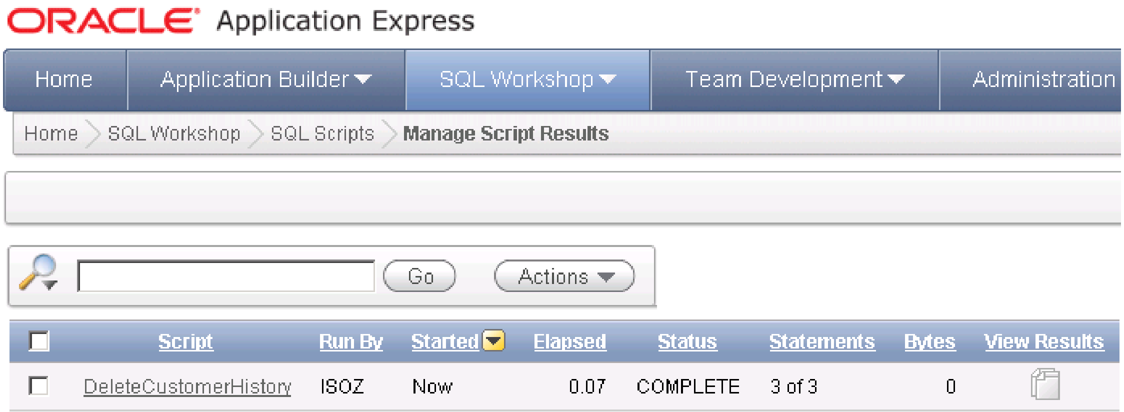
We then give a name to the script in the field **Script Name** and we write our script (it's a little bit false because it's simplified for pedagogical reasons!!):



and then we click on **Run**:



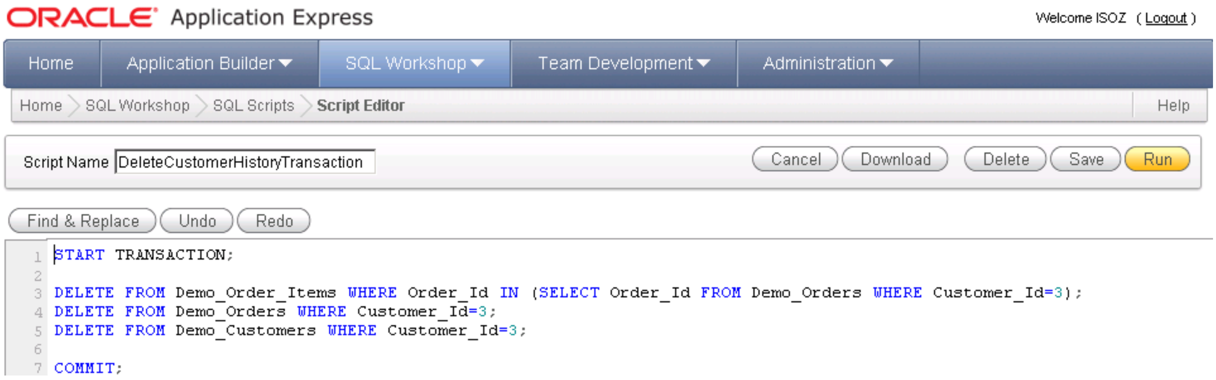
and then on **Run Now**:



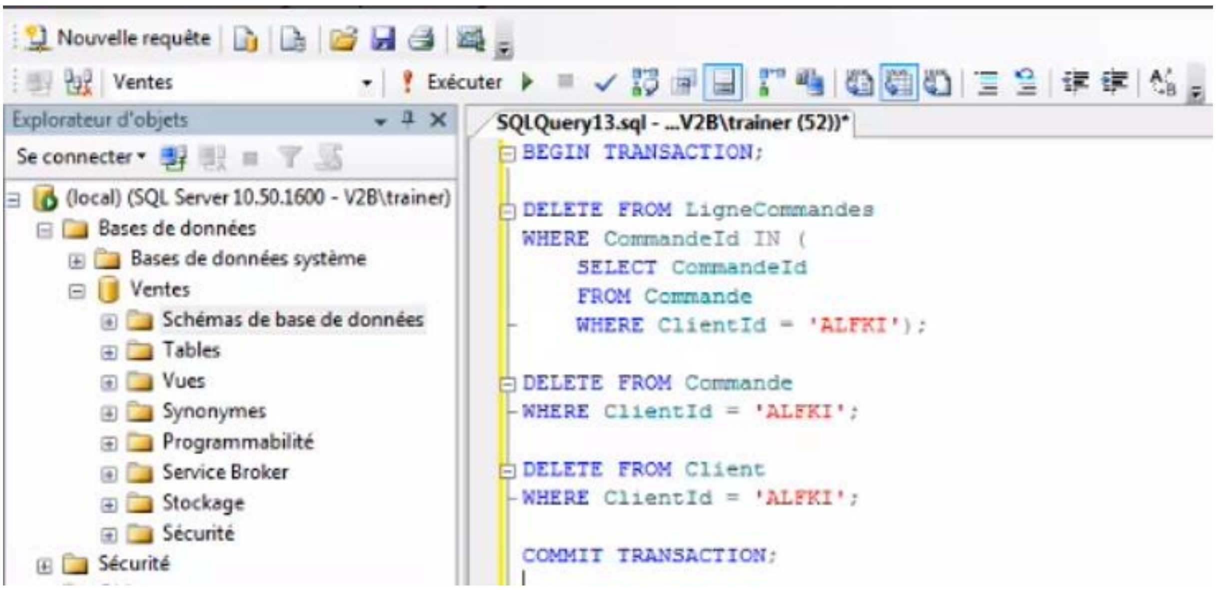
It works for sure **but in reality, a lot of troubles can occur!** During the deletion for example:

- The database can crash or be shut down for maintenance
- Some user could change some sensible data
- or any other know and imaginable error...

Here is an example where we want to be sure to delete everything related to customer 3:



in SQL Server the T-SQL syntax is similar:



To make the changes done in a transaction permanent use the COMMIT statement.

As you can see below, if there is for example an error in the script, the beginning will be committed (row 2) but the rest won't be executed:

Number	Elapsed	Statement	Feedback	Rows
1	0.08	START TRANSACTION	ERR-1001 Script not found.	-
2	0.44	DELETE FROM Demo_Order_Items WHERE Order_Id IN (SELECT Order	11 row(s) deleted.	11
3	0.00	DELETE FORM Demo_Orders WHERE Customer_Id=3	ORA-00942: table or view does not exist	-
4	0.10	DELETE FROM Demo_Customers WHERE Customer_Id=3	ORA-02292: integrity constraint (ISOZ.DEMO_ORDERS_CUSTOMER_ID_FK) violated - child record found	-
5	0.02	COMMIT	Statement processed.	0
			row(s) 1 - 5 of 5	

Download

Statements Processed 5
Successful 2
With Errors 3

This is very dangerous! That's why we will see the RoolBack.

9.3.1.1.2 Simple ROLLBACK Example

To roll back the changes done in a transaction give rollback statement. Rollback restore the state of the database to the last commit point.

We take the same script a before but we change it a little bit:

Script Name RollbackTransactionDemo   Cancel Download Delete Save Run

Find & Replace Undo Redo

```
START TRANSACTION;

DELETE FROM Demo_Order_Items WHERE Order_Id IN (SELECT Order_Id FROM Demo_Orders WHERE Customer_Id=5);
DELETE FROM Demo_Orders WHERE Customer_Id=5;
DELETE FROM Demo_Customers WHERE Customer_Id=5;

ROLLBACK;
```

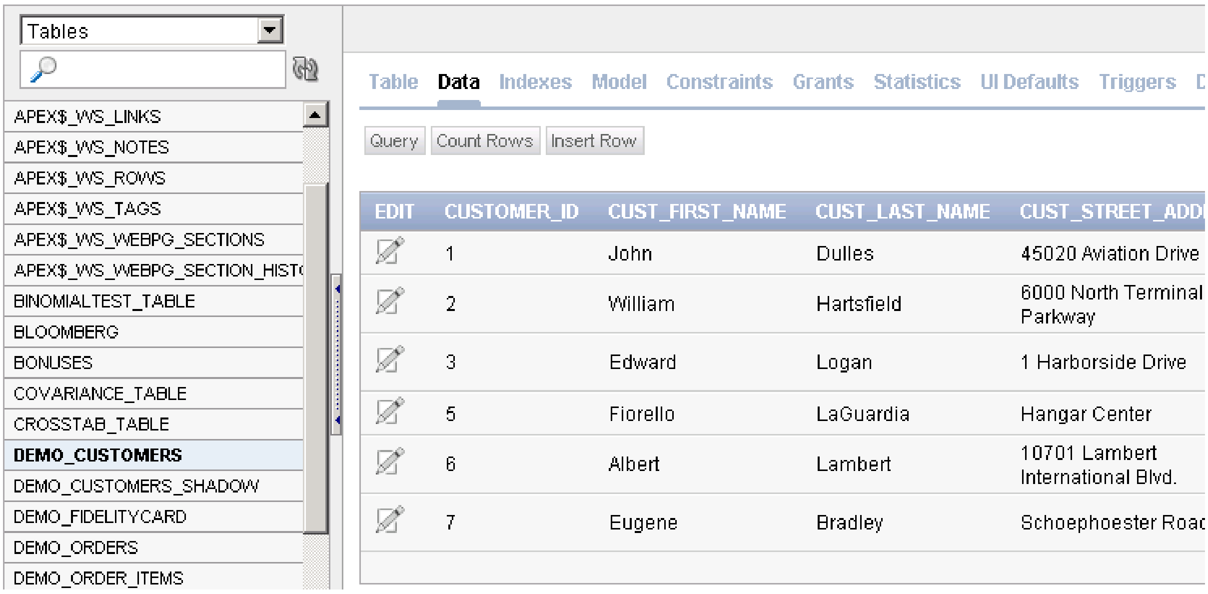
Now if we run it we can see in the log:

Number	Elapsed	Statement	Feedback	Rows
1	0.02	START TRANSACTION	ERR-1001 Script not found.	-
2	0.03	DELETE FROM Demo_Order_Items WHERE Order_Id IN (SELECT Order	5 row(s) deleted.	5
3	0.03	DELETE FROM Demo_Orders WHERE Customer_Id=5	1 row(s) deleted.	1
4	0.01	DELETE FROM Demo_Customers WHERE Customer_Id=5	1 row(s) deleted.	1
5	0.02	ROLLBACK	Statement processed.	0
			row(s) 1 - 5 of 5	

Download

Statements Processed 5
Successful 4
With Errors 1

but because of the ROLLBACK, everything that is related to the customer 5 is still here in reality:



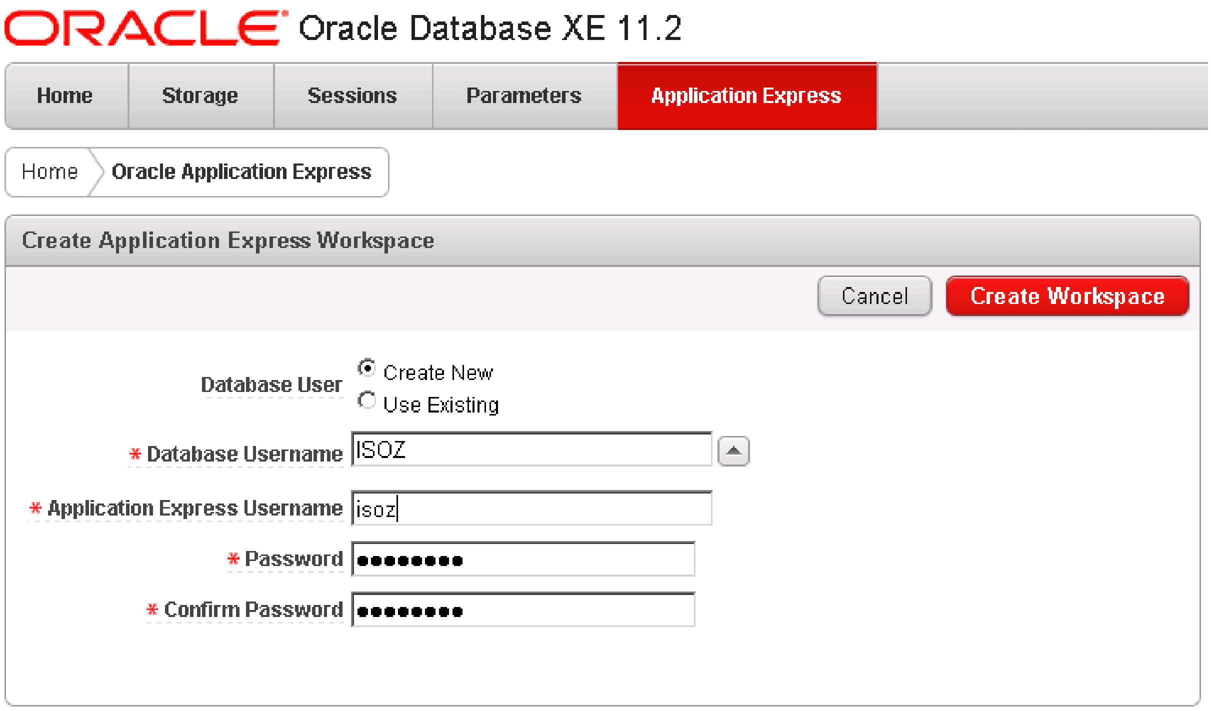
9.3.1.1.3 LOCK et UNLOCK

MySQL enables client sessions to acquire table locks explicitly for the purpose of cooperating with other sessions for access to tables, or to prevent other sessions from modifying tables during periods when a session requires exclusive access to them. A session can acquire or release locks only for itself. One session cannot acquire locks for another session or release locks held by another session.

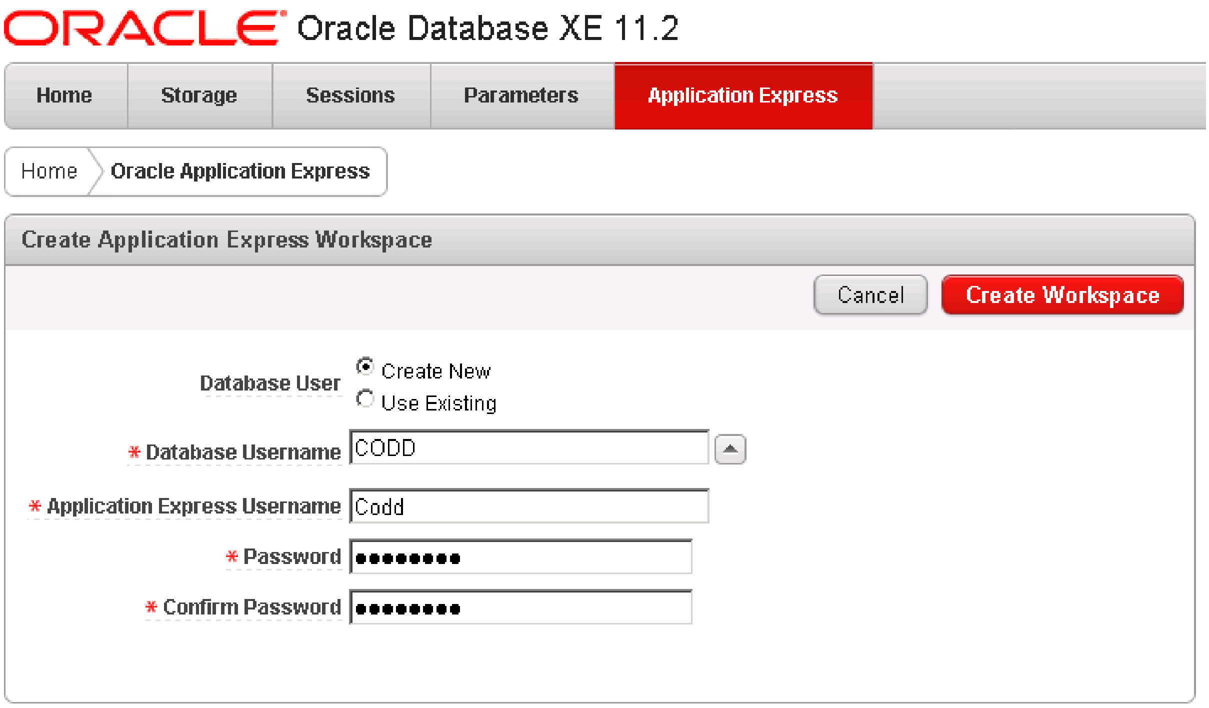
Locks may be used to emulate transactions or to get more speed when updating tables. This is explained in more detail later in this section.

LOCK TABLES explicitly acquire table locks **for the current client session**!

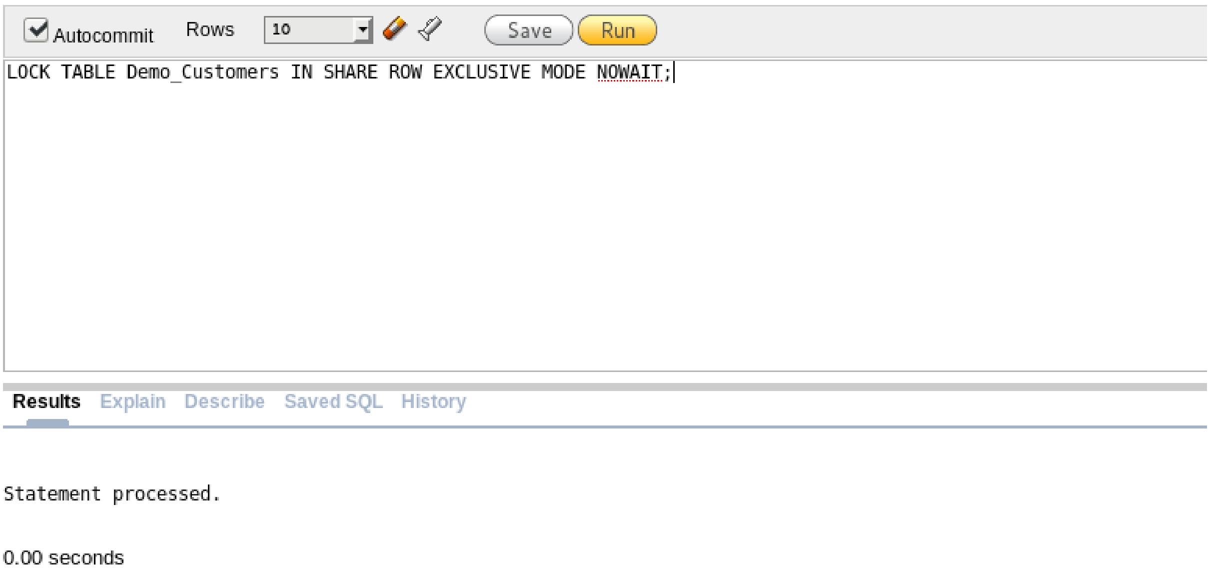
For the example, remember that at page 277 we have created two sessions:



And:



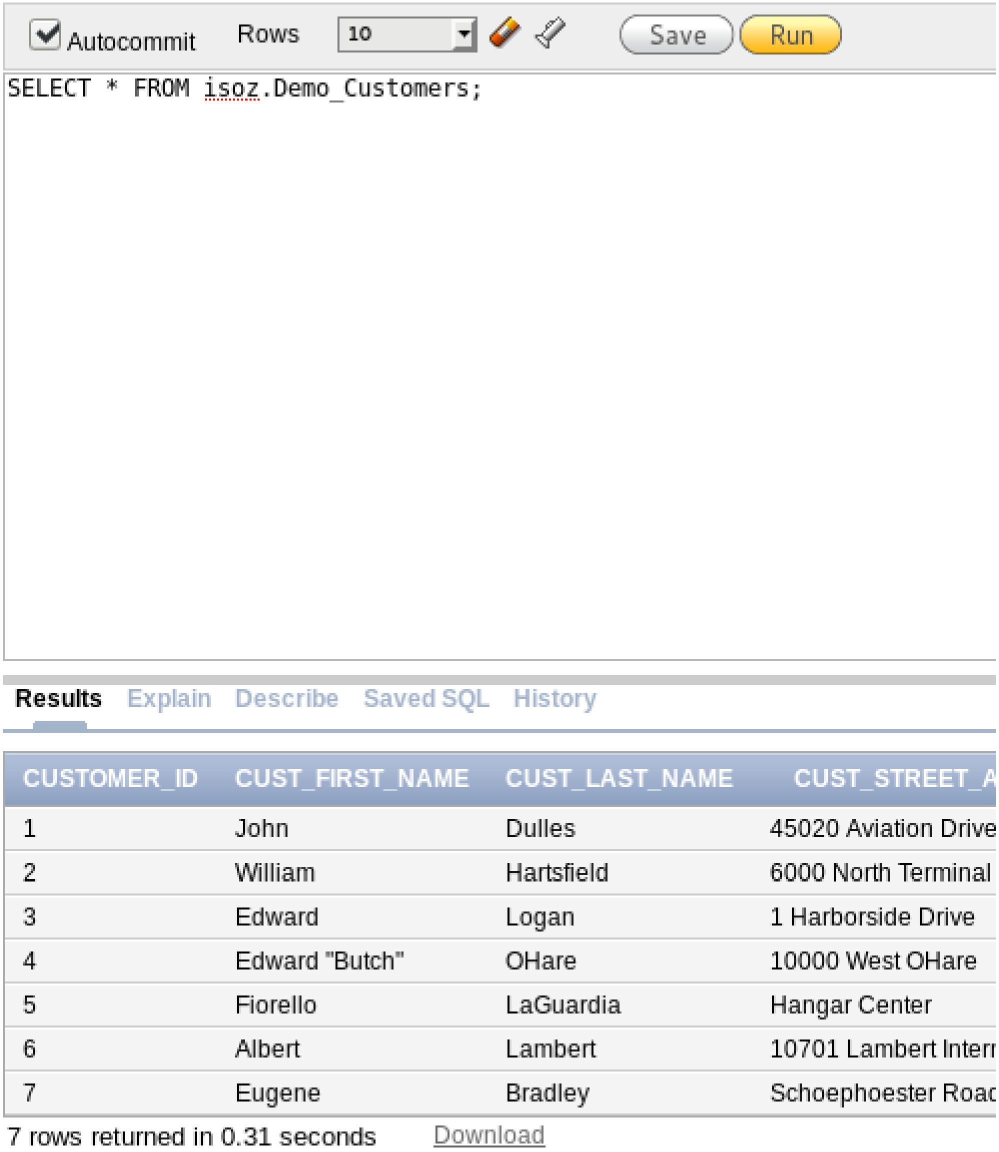
From the ISOZ session, we lock on table immediately:



lock_mode	Explanation
ROW SHARE	Allows concurrent access to the table, but users are prevented from locking the entire table for exclusive access.
ROW EXCLUSIVE	Allows concurrent access to the table, but users are prevented from locking the entire table with exclusive access and locking the table in share mode.
SHARE UPDATE	Allows concurrent access to the table, but users are prevented from locking the entire table for exclusive access.
SHARE	Allows concurrent queries but users are prevented from updating the locked table.
SHARE ROW EXCLUSIVE	Users can view records in table, but are prevented from updating the table or from locking the table in SHARE mode.
EXCLUSIVE	Allows queries on the locked table, but no other activities.

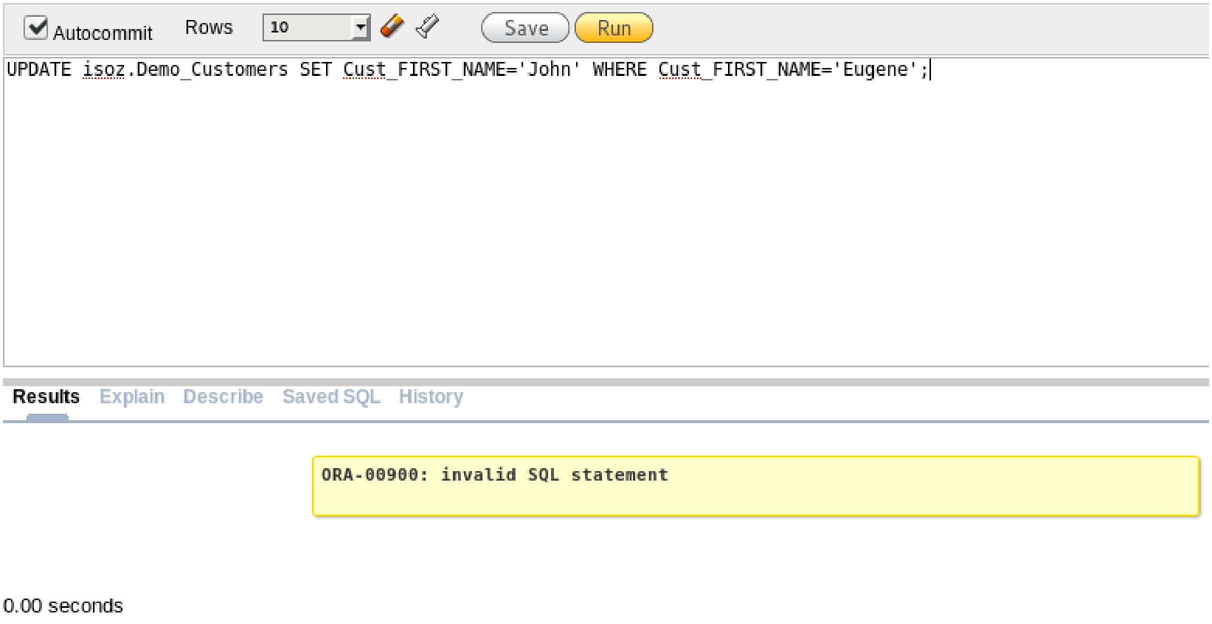
Table 1 Types of lock modes

We go to the CODD session afterwards and run first a simple query:

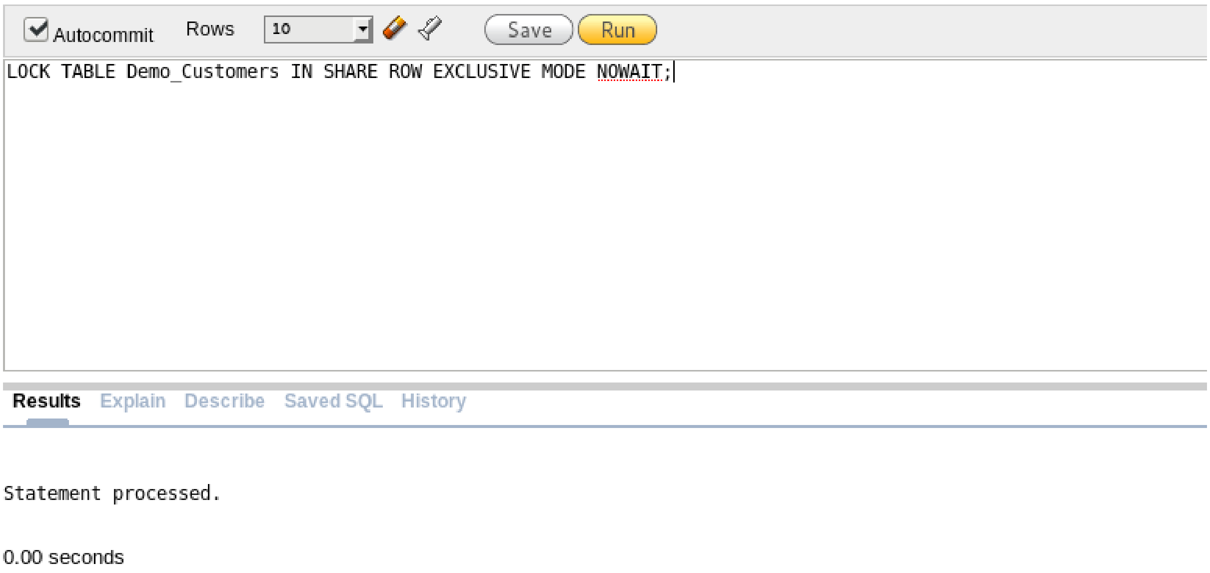


CUSTOMER_ID	CUST_FIRST_NAME	CUST_LAST_NAME	CUST_STREET_A
1	John	Dulles	45020 Aviation Drive
2	William	Hartsfield	6000 North Terminal
3	Edward	Logan	1 Harborside Drive
4	Edward "Butch"	OHare	10000 West OHare
5	Fiorello	LaGuardia	Hangar Center
6	Albert	Lambert	10701 Lambert Interr
7	Eugene	Bradley	Schoephoester Roac

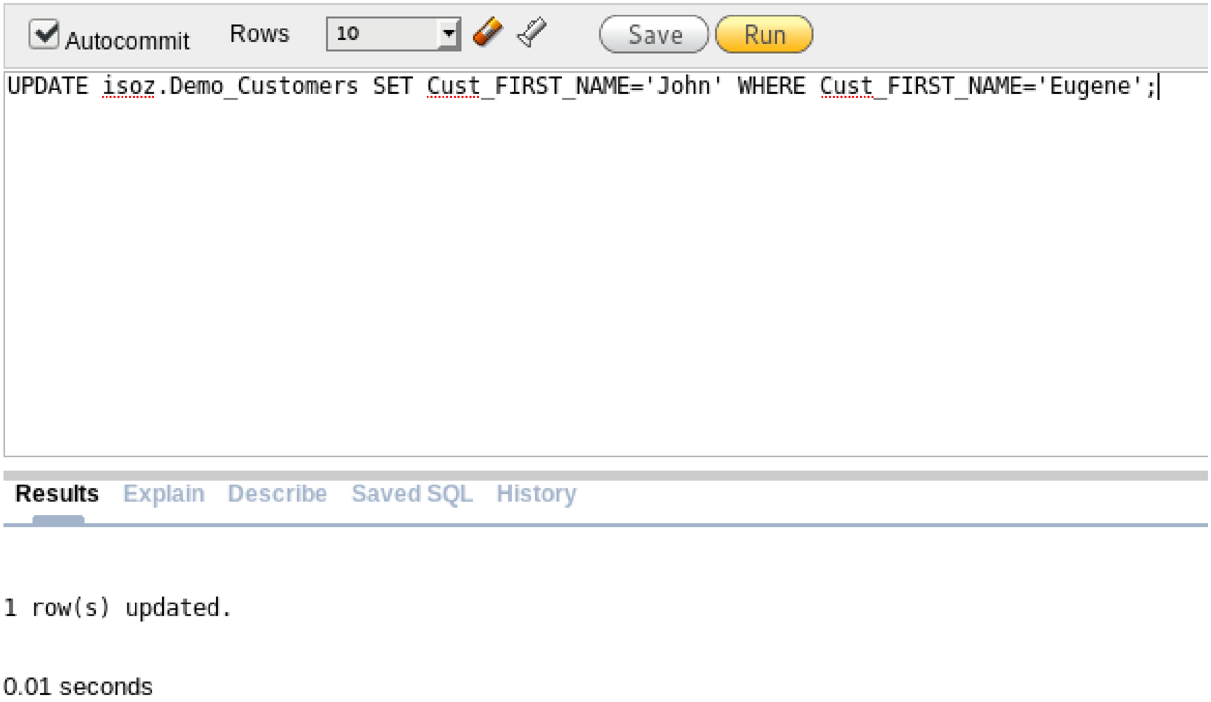
But let's see if we can update a row for exemple:



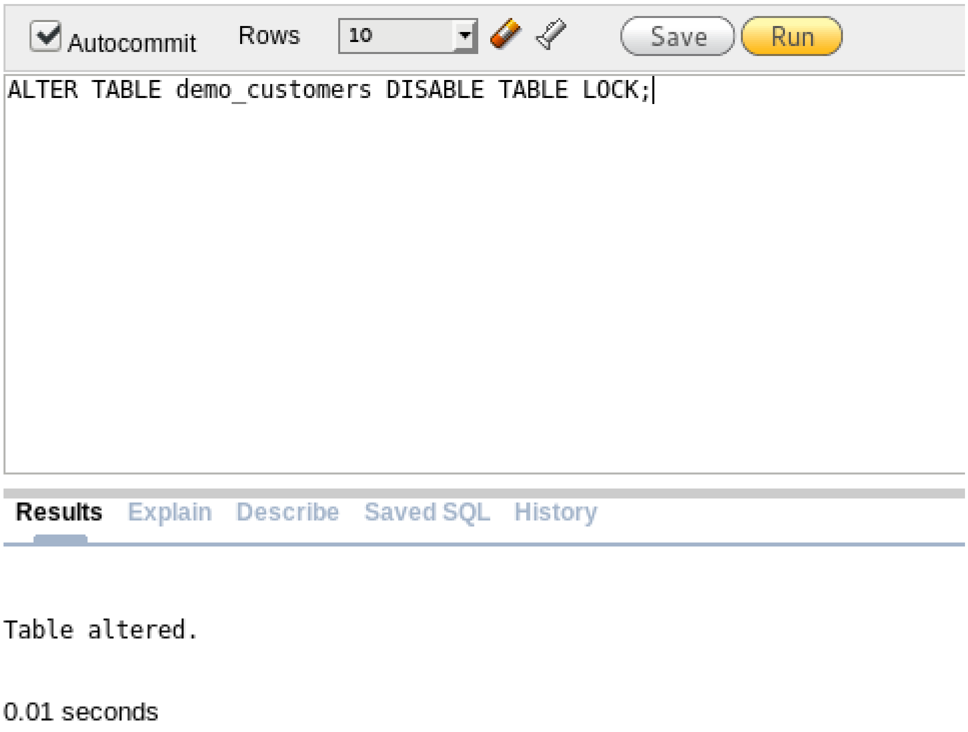
If we unlock the table in the ISOZ session a bit:



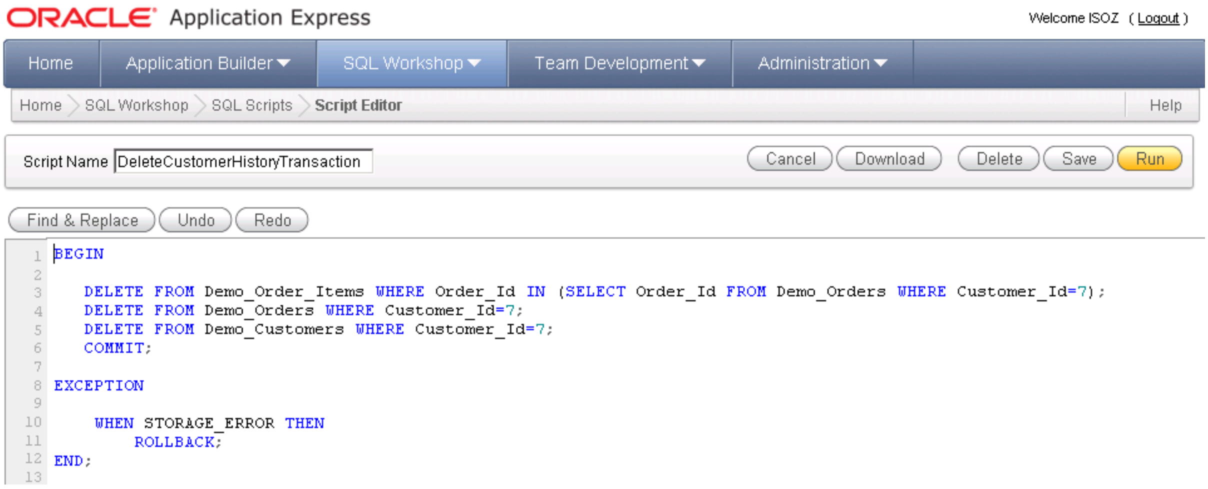
Then Codd can run the update:



And if we want to completely unlock the table:



9.3.2 TRANSACTION with EXCEPTION



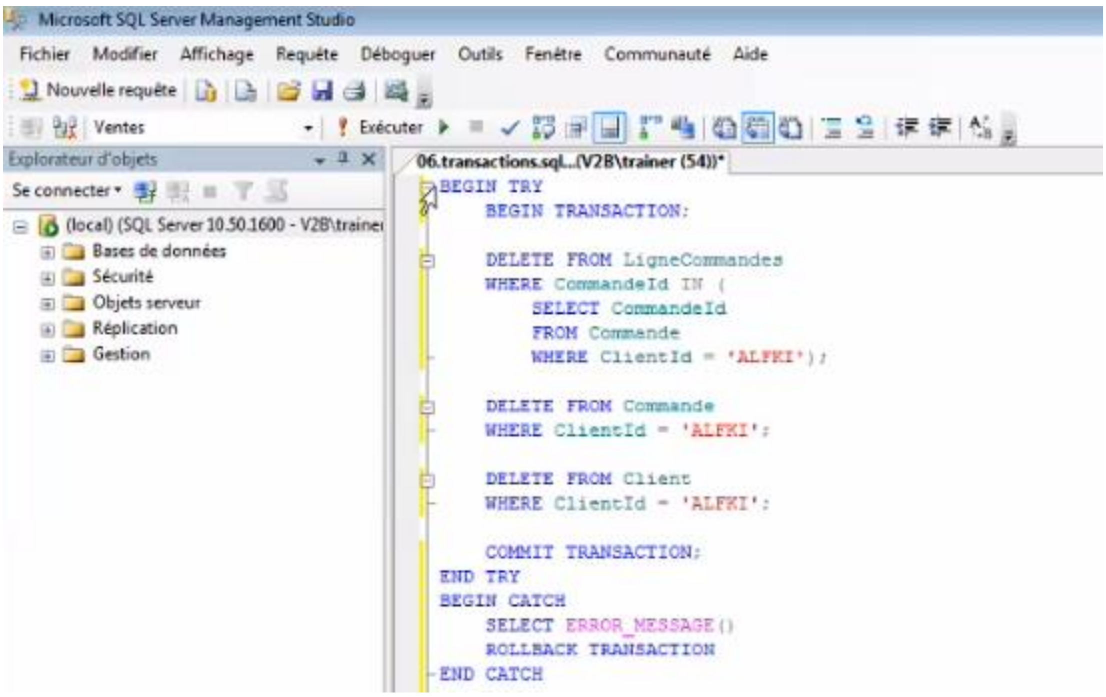
type of errors in Oracle:

Exception	Raised when ...
ACCESS_INTO_NULL	Your program attempts to assign values to the attributes of an uninitialized (atomically null) object.
CASE_NOT_FOUND	None of the choices in the WHEN clauses of a CASE statement is selected, and there is no ELSE clause.
COLLECTION_IS_NULL	Your program attempts to apply collection methods other than EXISTS to an uninitialized (atomically null) nested table or varray, or the program attempts to assign values to the elements of an uninitialized nested table or varray.
CURSOR_ALREADY_OPEN	Your program attempts to open an already open cursor. A cursor must be closed before it can be reopened. A cursor FOR loop automatically opens the cursor to which it refers. So, your program cannot open that cursor inside the loop.
DUP_VAL_ON_INDEX	Your program attempts to store duplicate values in a database column that is constrained by a unique index.
INVALID_CURSOR	Your program attempts an illegal cursor operation such as closing an unopened cursor.
INVALID_NUMBER	In a SQL statement, the conversion of a character string into a number fails because the string does not represent a valid number. (In procedural statements, VALUE_ERROR is raised.) This exception is also raised when the LIMIT-clause expression in a bulk FETCH statement does not evaluate to a positive number.
LOGIN_DENIED	Your program attempts to log on to Oracle with an invalid username and/or password.
NO_DATA_FOUND	A SELECT INTO statement returns no rows, or your program references a deleted element in a nested table or an uninitialized element in an index-by table. SQL aggregate functions such as AVG and SUM always return a value or a null. So, a SELECT INTO statement that calls an aggregate function never raises NO_DATA_FOUND. The FETCH statement is expected to return no rows eventually, so when that happens, no exception is raised.
NOT_LOGGED_ON	Your program issues a database call without being connected to Oracle.

Exception	Raised when ...
PROGRAM_ERROR	PL/SQL has an internal problem.
ROWTYPE_MISMATCH	The host cursor variable and PL/SQL cursor variable involved in an assignment have incompatible return types. For example, when an open host cursor variable is passed to a stored subprogram, the return types of the actual and formal parameters must be compatible.
SELF_IS_NULL	Your program attempts to call a MEMBER method on a null instance. That is, the built-in parameter SELF (which is always the first parameter passed to a MEMBER method) is null.
STORAGE_ERROR	PL/SQL runs out of memory or memory has been corrupted.
SUBSCRIPT_BEYOND_COUNT	Your program references a nested table or varray element using an index number larger than the number of elements in the collection.
SUBSCRIPT_OUTSIDE_LIMIT	Your program references a nested table or varray element using an index number (-1 for example) that is outside the legal range.
SYS_INVALID_ROWID	The conversion of a character string into a universal rowid fails because the character string does not represent a valid rowid.
TIMEOUT_ON_RESOURCE	A time-out occurs while Oracle is waiting for a resource.
TOO_MANY_ROWS	A SELECT INTO statement returns more than one row.
VALUE_ERROR	An arithmetic, conversion, truncation, or size-constraint error occurs. For example, when your program selects a column value into a character variable, if the value is longer than the declared length of the variable, PL/SQL aborts the assignment and raises VALUE_ERROR. In procedural statements, VALUE_ERROR is raised if the conversion of a character string into a number fails. (In SQL statements, INVALID_NUMBER is raised.)
ZERO_DIVIDE	Your program attempts to divide a number by zero.

Tableau 12 Oracle typical errors

in SQL Server the T-SQL syntax is almost very different:

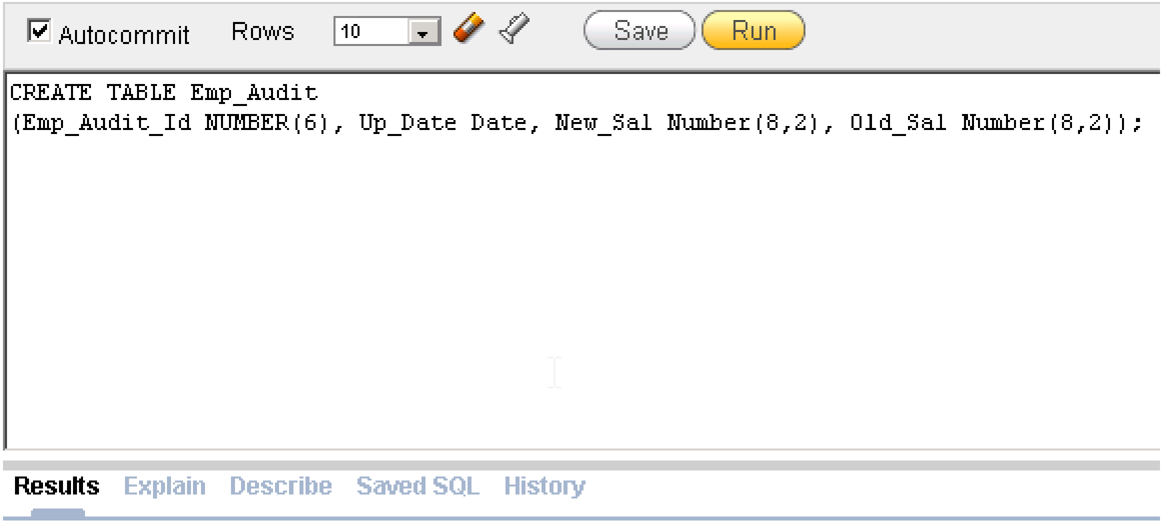


9.4 Triggers

Like a stored procedure, a trigger is a named PL/SQL unit that is stored in the database and can be invoked repeatedly. Unlike a stored procedure, you can enable and disable a trigger, but you cannot explicitly invoke it. While a trigger is enabled, the database automatically invokes it—that is, the trigger fires—whenever its triggering event occurs. While a trigger is disabled, it does not fire.

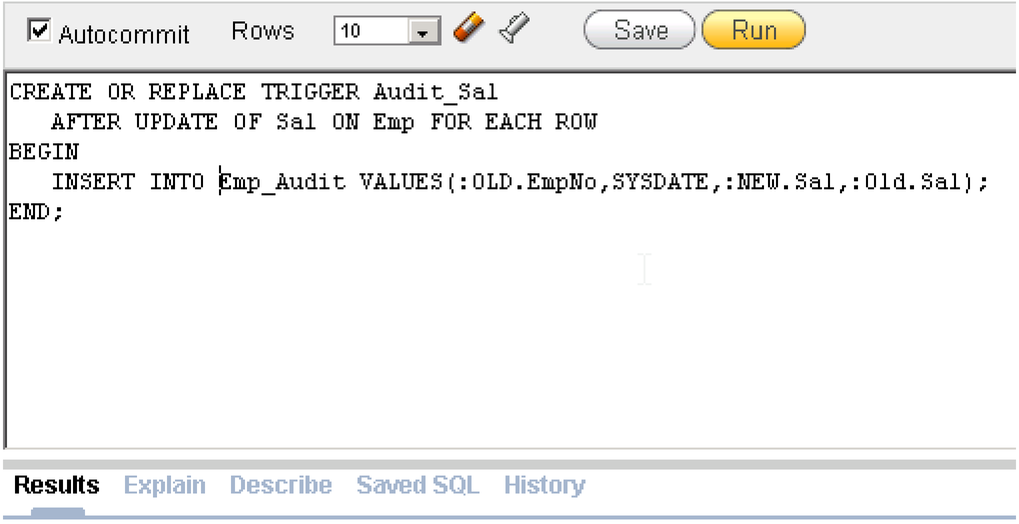
You create a trigger with the CREATE TRIGGER statement. You specify the triggering event in terms of triggering statements and the item on which they act. The trigger is said to be created on or defined on the item, which is either a table, a view, a schema, or the database. You also specify the timing point, which determines whether the trigger fires before or after the triggering statement runs and whether it fires for each row that the triggering statement affects.

To see an easy example first create the following table:



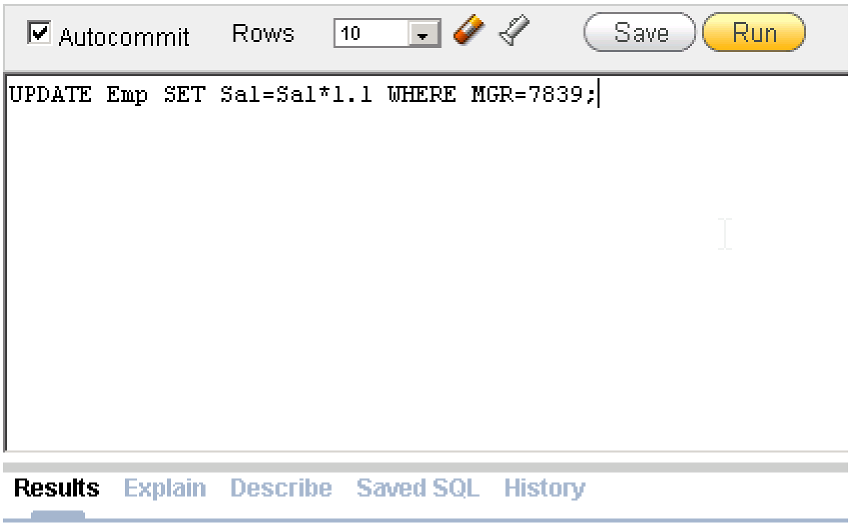

```
Table created.
```

Then run the following code to create the trigger:



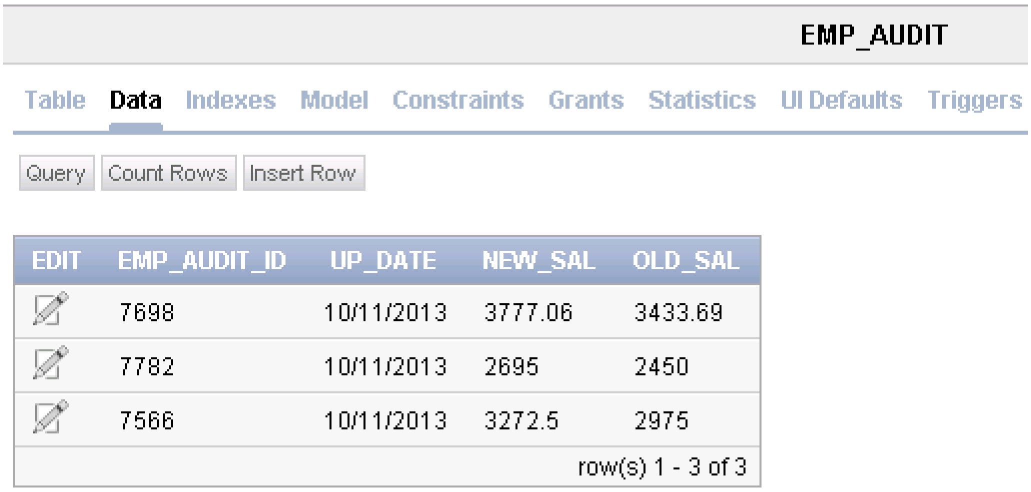

```
Trigger created.
```

Then fire the trigger with for example the following code:




```
3 row(s) updated.
```

You will then have:



EDIT	EMP_AUDIT_ID	UP_DATE	NEW_SAL	OLD_SAL
	7698	10/11/2013	3777.06	3433.69
	7782	10/11/2013	2695	2450
	7566	10/11/2013	3272.5	2975

row(s) 1 - 3 of 3

get it! ;-)

10 SQL Tutorial for Injection (hacking)

10.1 SQL Injection Based on ""="" is Always True

Here is a common construction, used to verify user login to a web site:

User Name:

Password:

Imagine that the Server Code is:

```
uName = getRequestString("UserName");
uPass = getRequestString("UserPass");

sql = "SELECT * FROM Users WHERE Name ='" + uName + "' AND Pass ='" + uPass
+ "'"
```

A smart hacker might get access to user names and passwords in a database by simply inserting " or ""=" into the user name or password text box.

The code at the server will create a valid SQL statement like this:

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

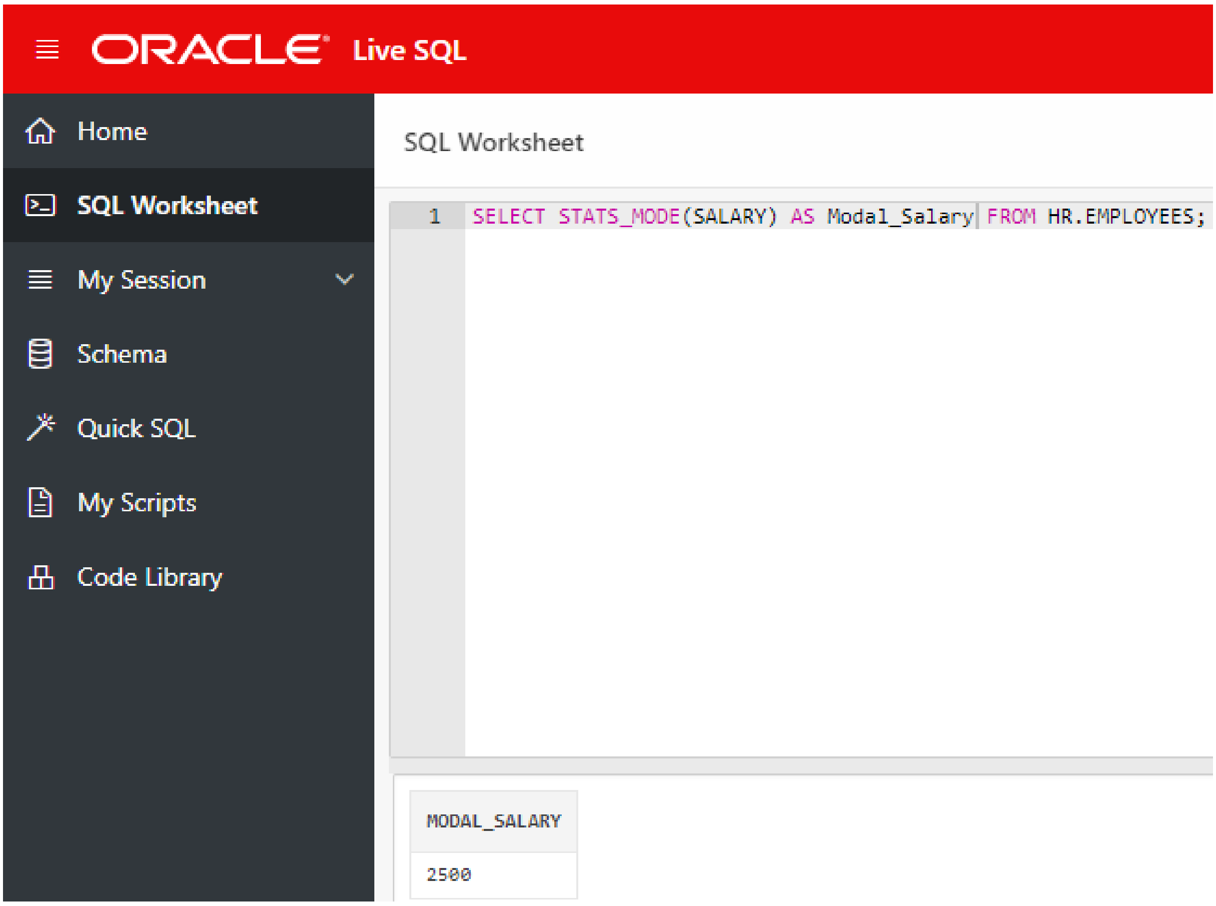
The result SQL is valid. It will return all rows from the table Users, since **WHERE ""=""** is always true.

11 SQL for Data Science

We will focus here on all STATS_XXXX functions of Oracle and only on these one. We have already introduced some of them earlier on page 189. We will repeat them here but also go more in deep in this subject and with ORACLE Live SQL!

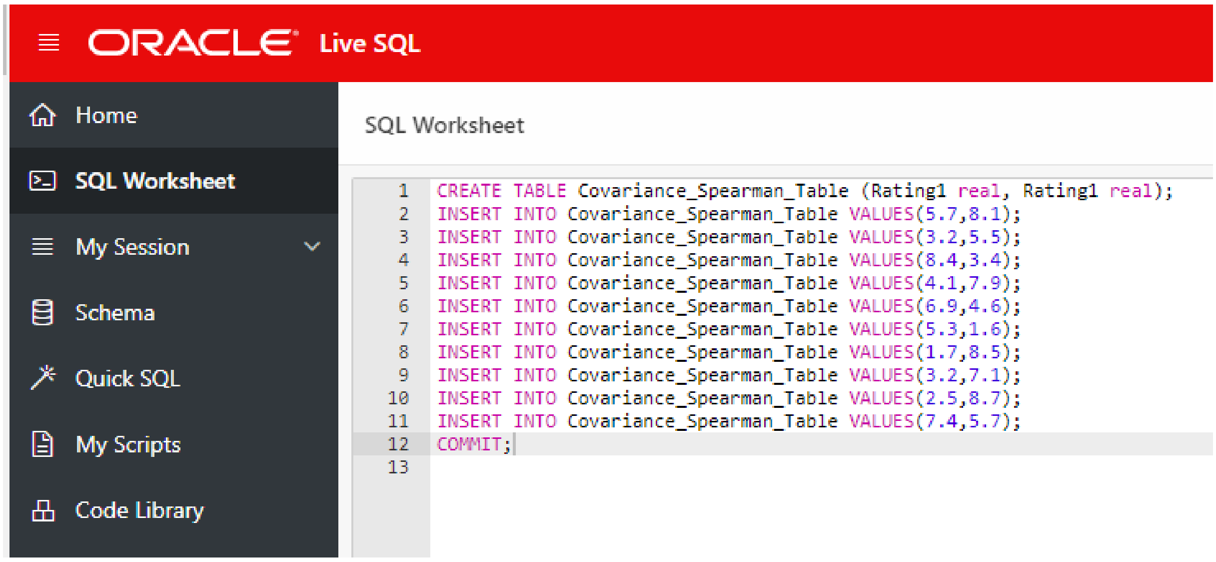
Let's start:

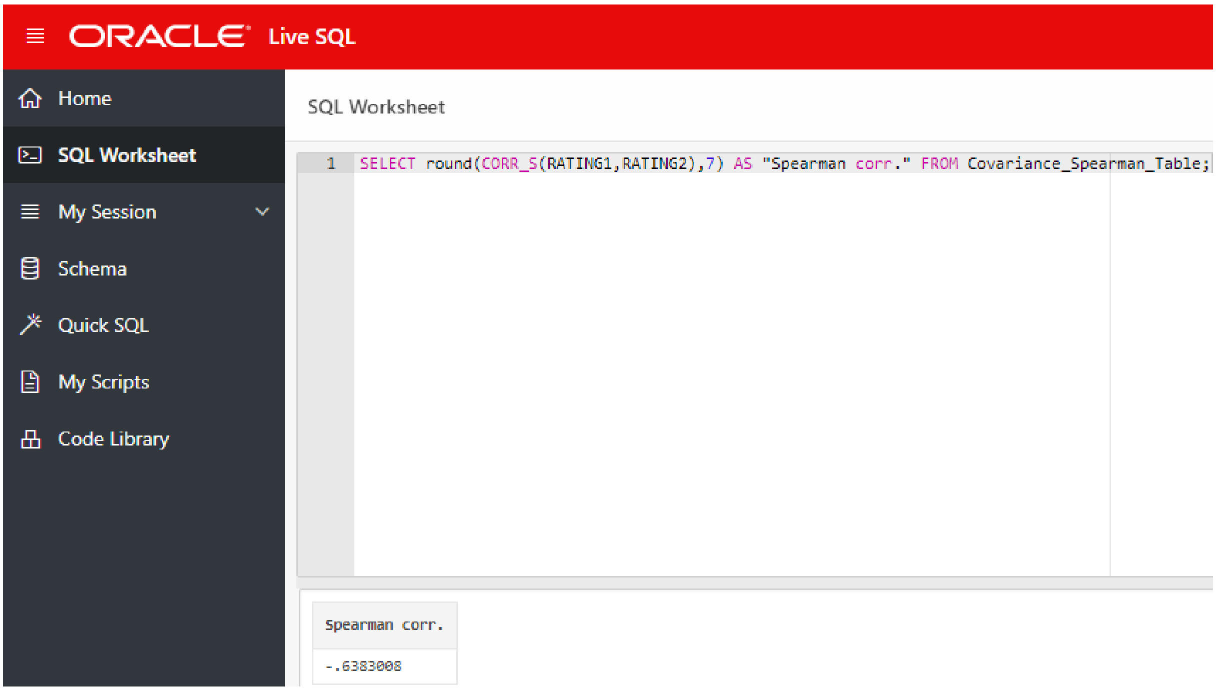
11.1 Modal Value

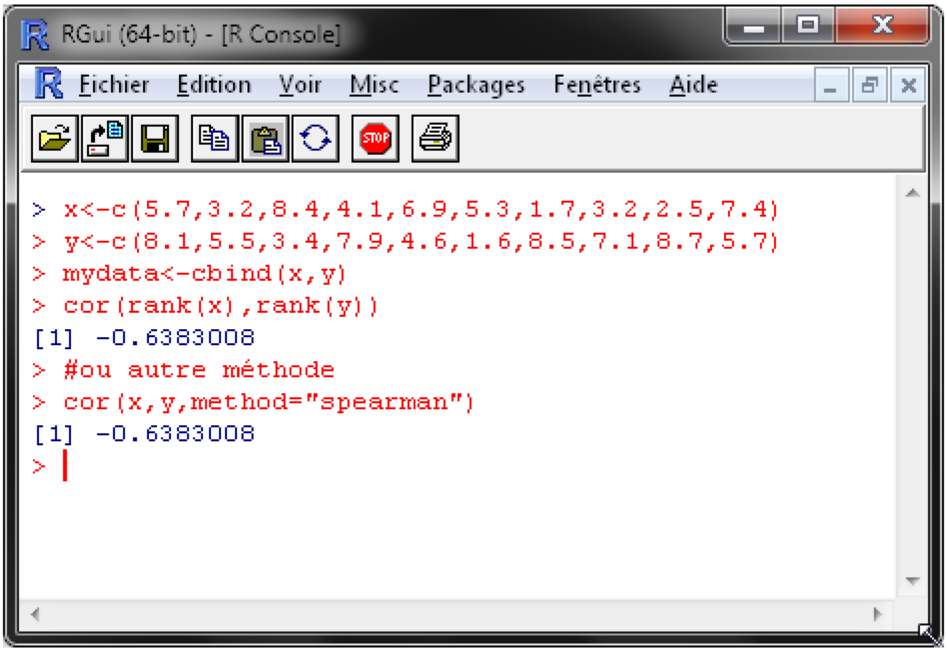


That's all what this function can do… hence…: No comments!

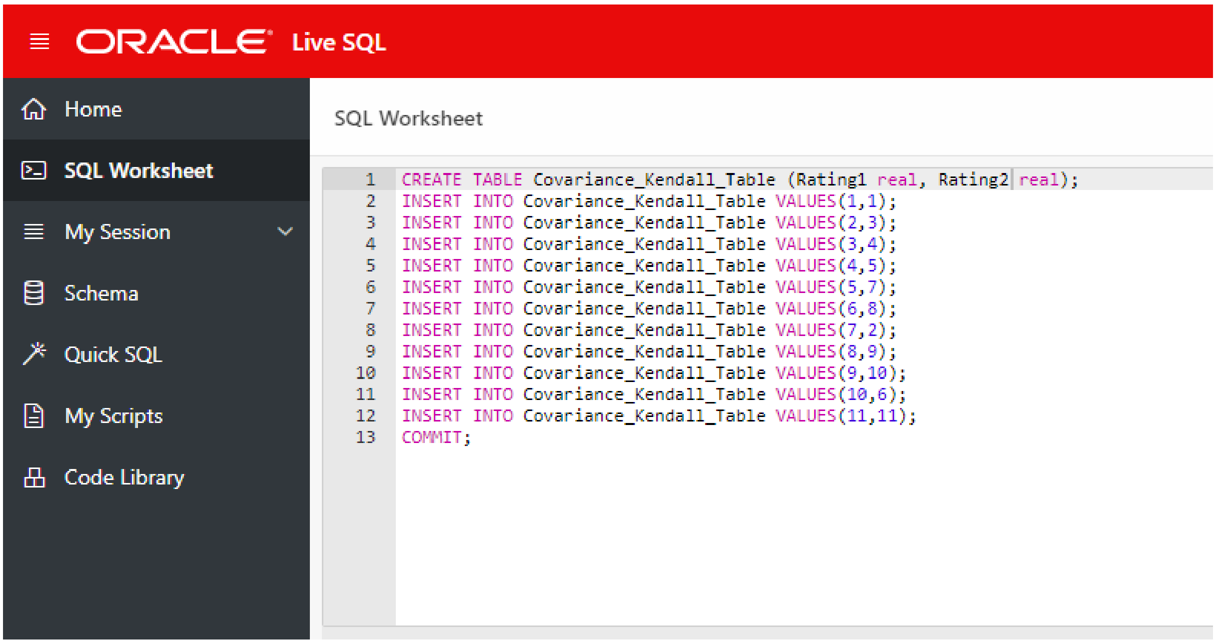
11.2 Spearman correlation coefficient

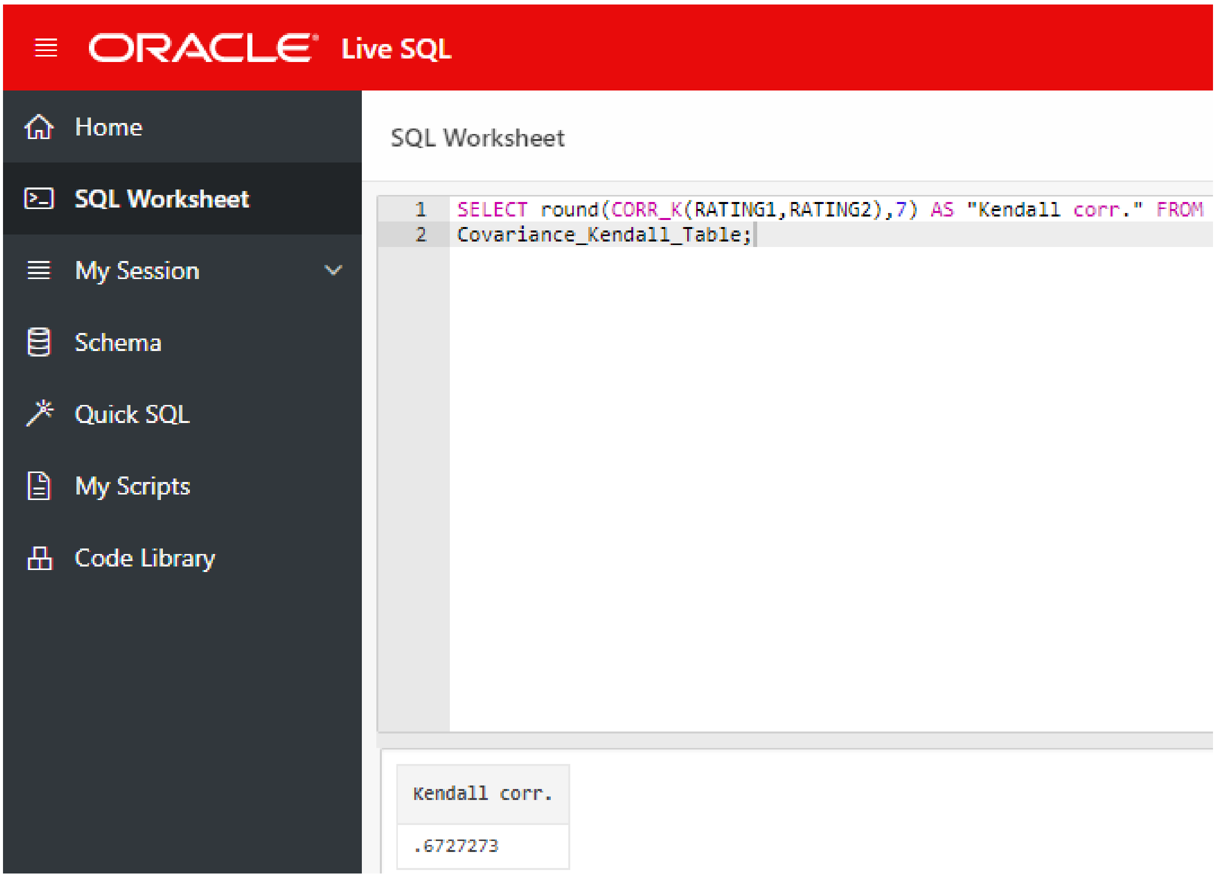


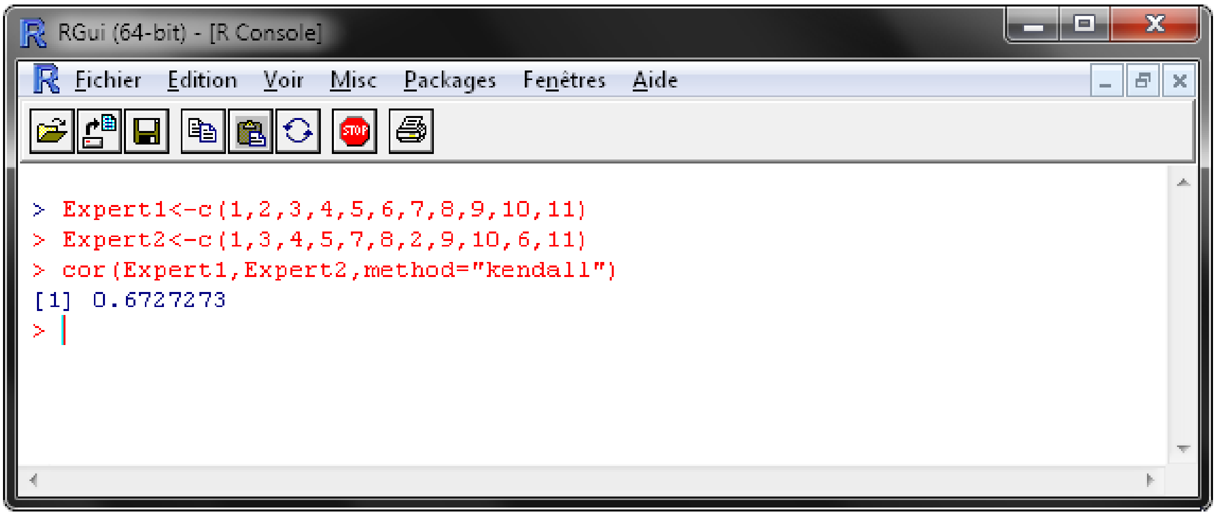



RGui (64-bit) - [R Console]
Fichier Edition Voir Misc Packages Fenêtres Aide
> x<-c(5.7,3.2,8.4,4.1,6.9,5.3,1.7,3.2,2.5,7.4)
> y<-c(8.1,5.5,3.4,7.9,4.6,1.6,8.5,7.1,8.7,5.7)
> mydata<-cbind(x,y)
> cor(rank(x),rank(y))
[1] -0.6383008
> #ou autre méthode
> cor(x,y,method="spearman")
[1] -0.6383008
>

11.3 Kendall correlation coefficient of concordance



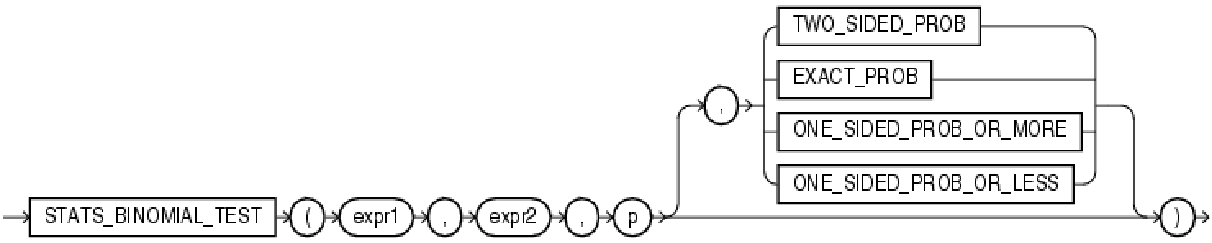



RGui (64-bit) - [R Console]
Fichier Edition Voir Misc Packages Fenêtres Aide
> Expert1<-c(1,2,3,4,5,6,7,8,9,10,11)
> Expert2<-c(1,3,4,5,7,8,2,9,10,6,11)
> cor(Expert1,Expert2,method="kendall")
[1] 0.6727273
>

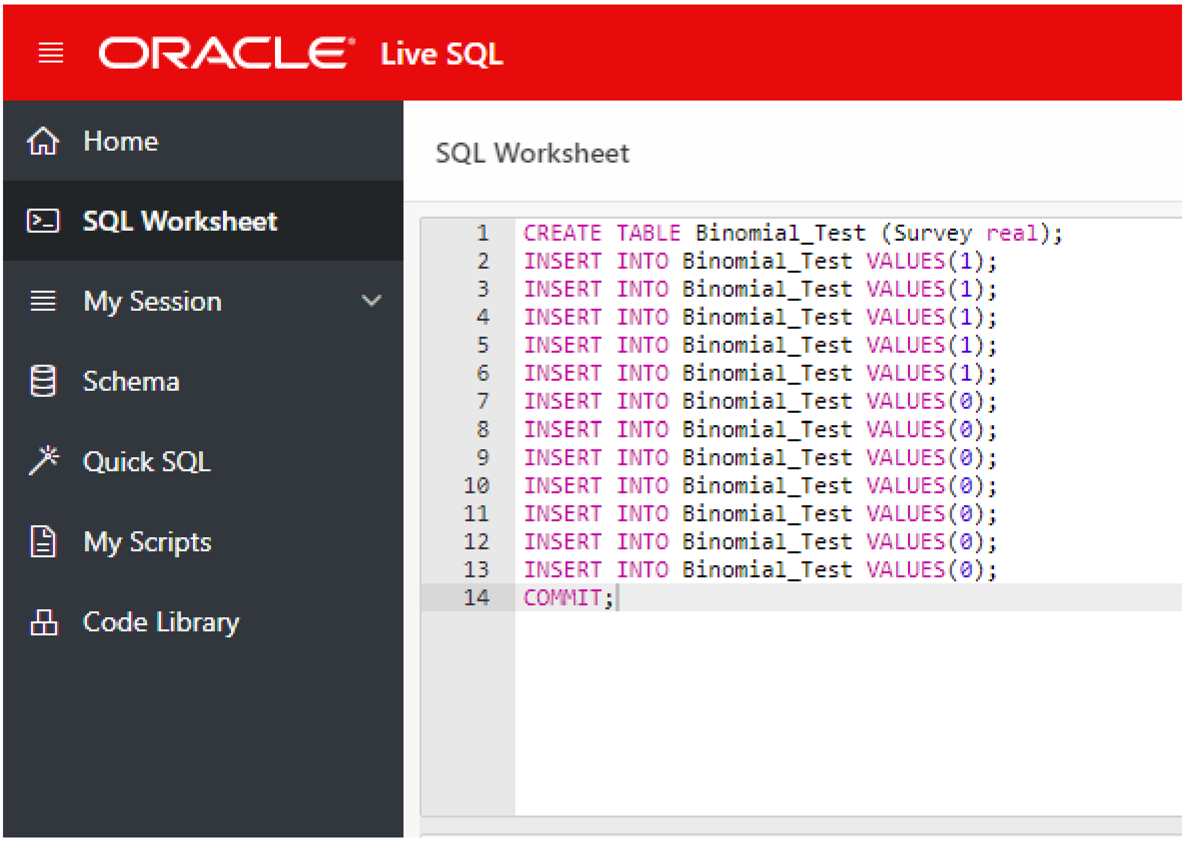
11.4 Binomial Probability

The name of this function STATS_BINOMIAL_TEST is really misleading! It's absolutely not a binomial test but just the cumulate probability function…

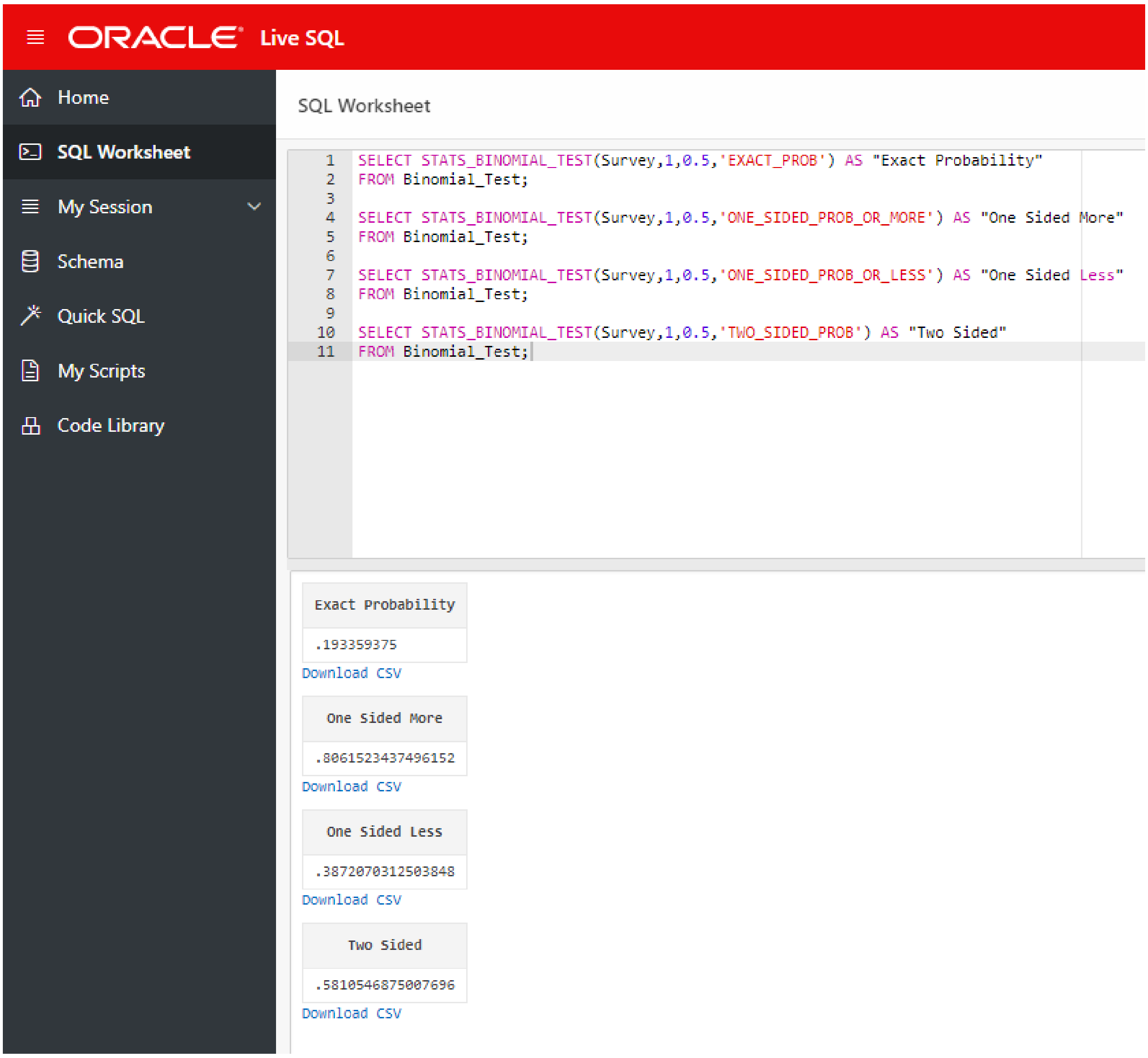
The function description is the following:



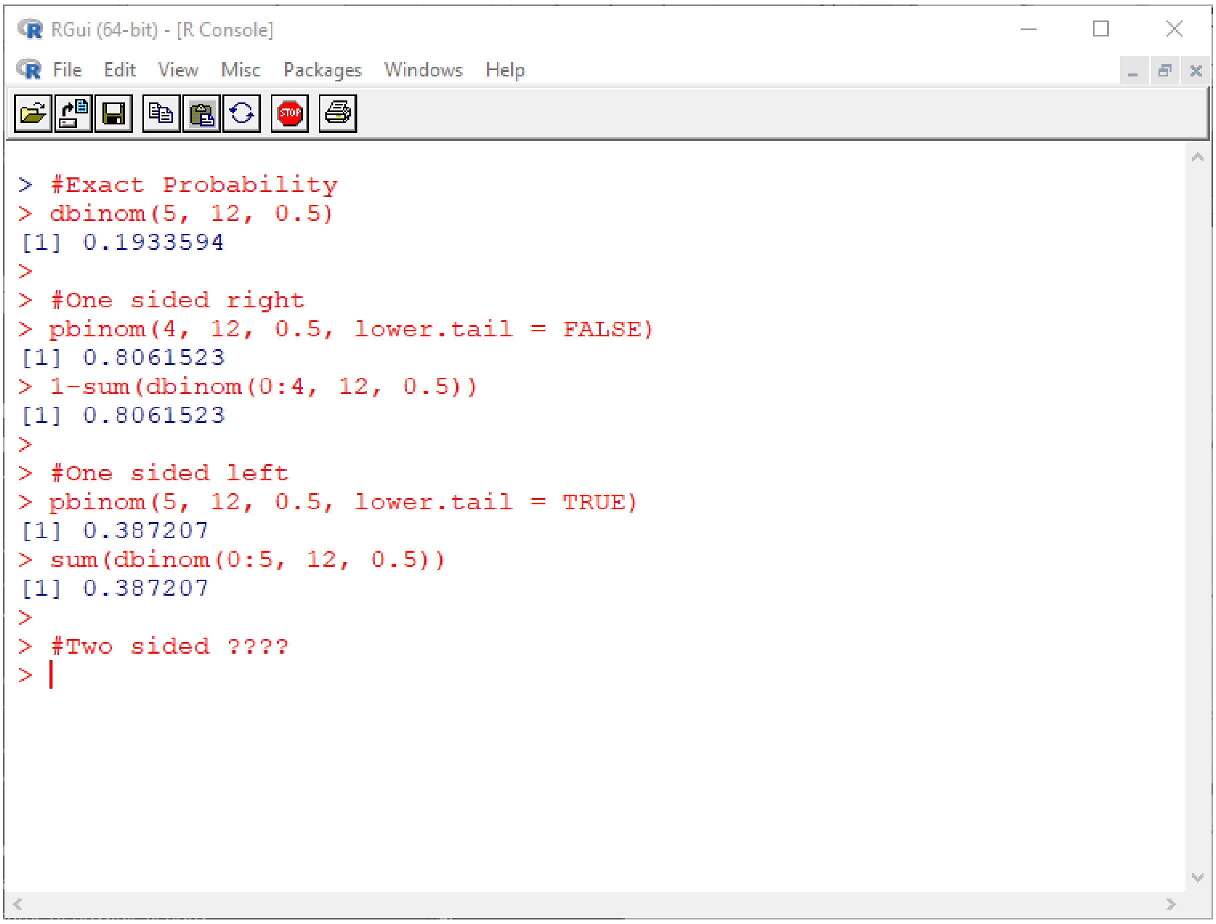
If we take the same data as in the theoretical training:



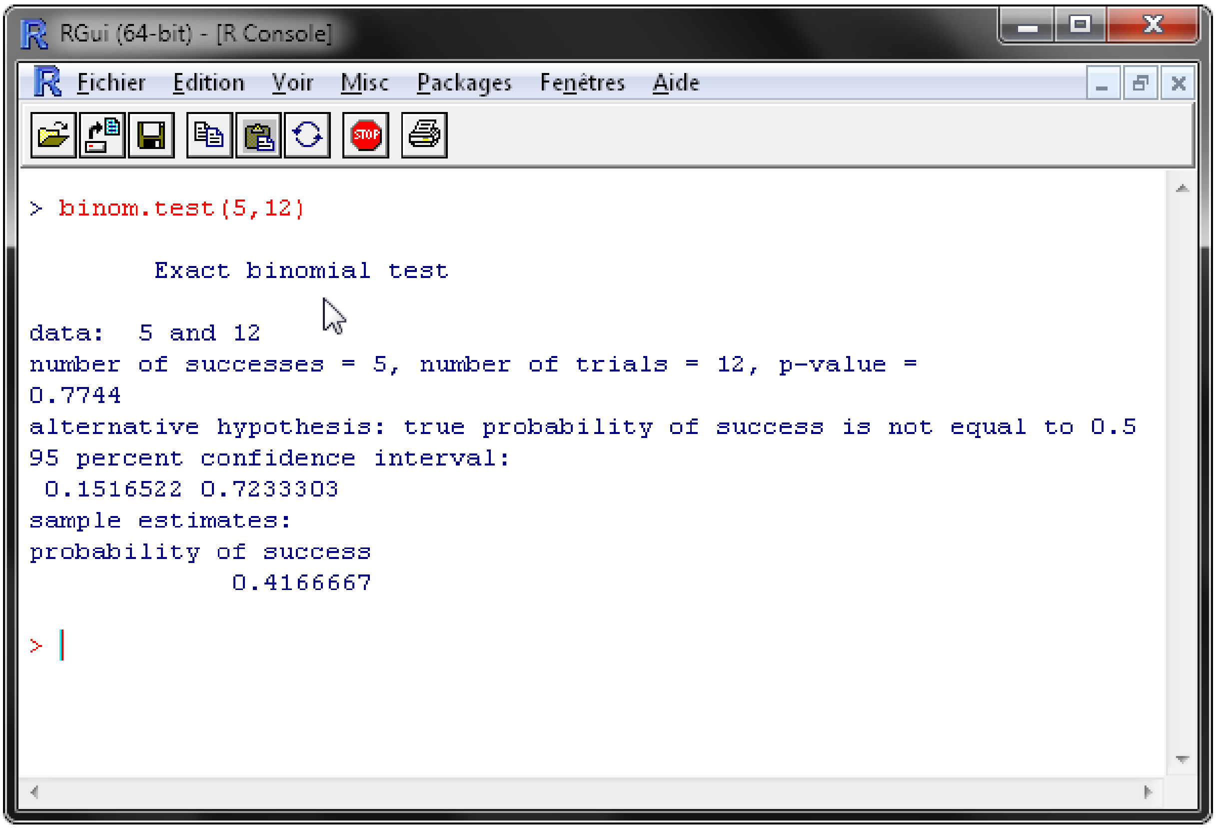
we get:



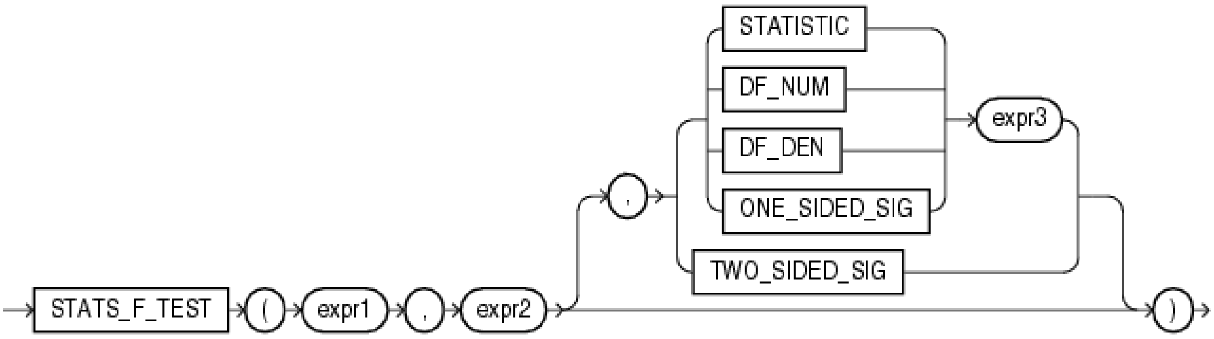
We can fall back on almost all results using the R statistical software (excepted that last one that I was not able to found how Oracle calculates it…):

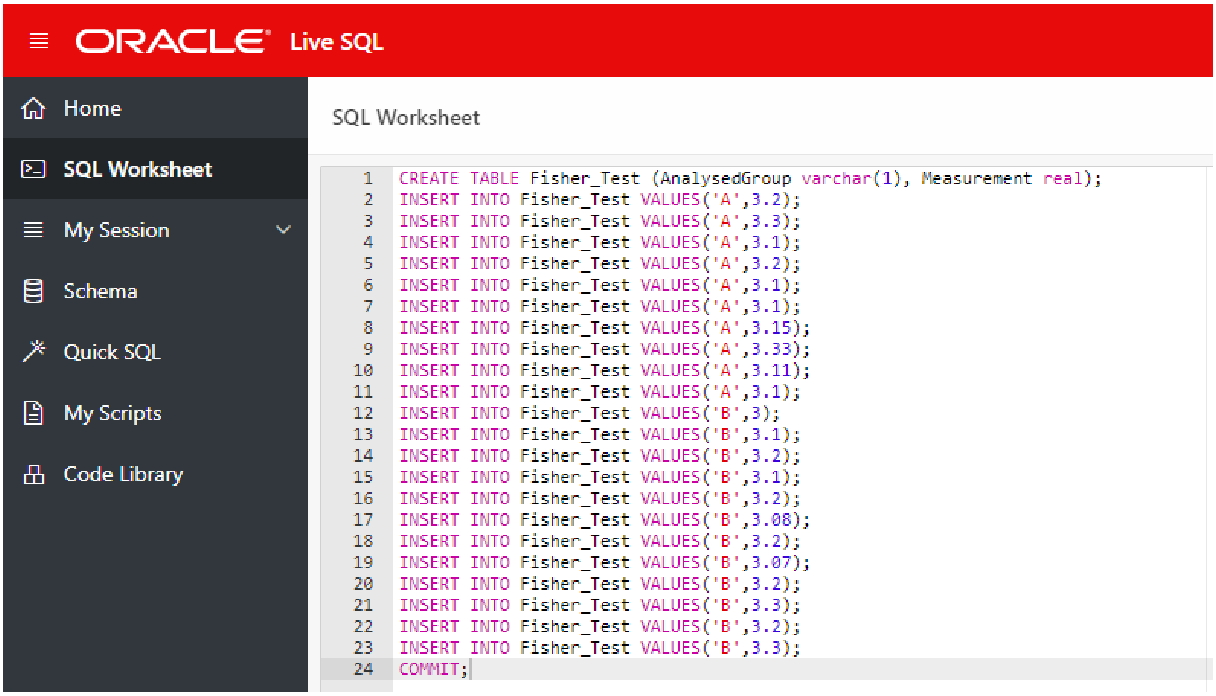


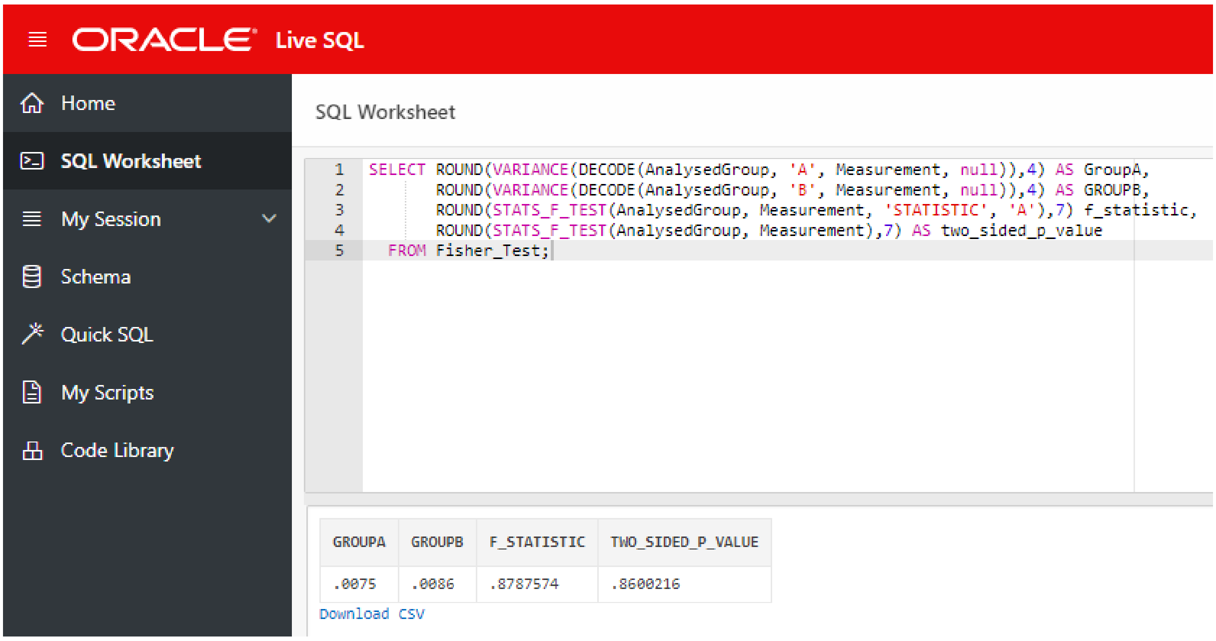
As you can see, this has nothing to do with a real binomial test:



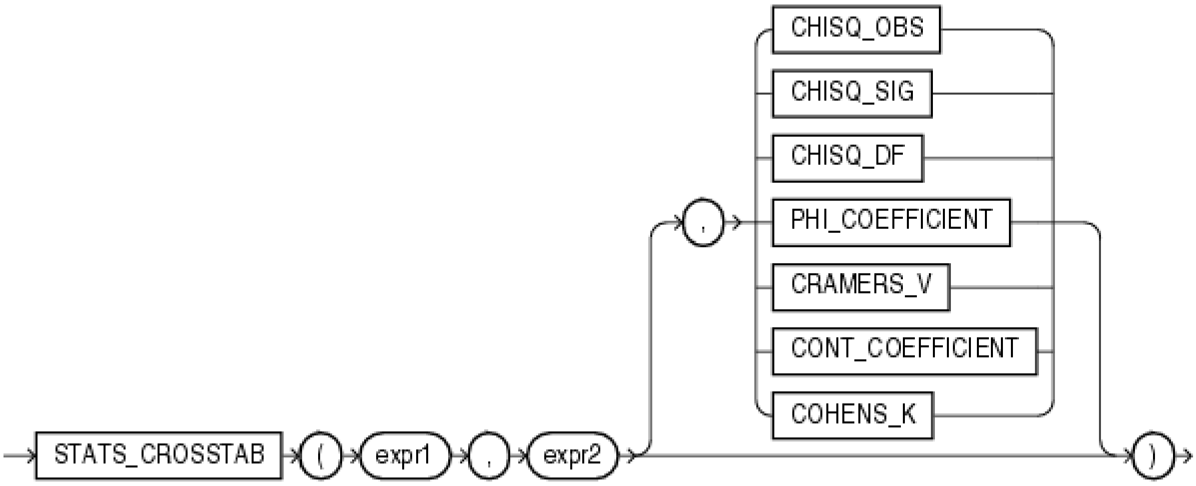
11.5 Fisher Variance Test

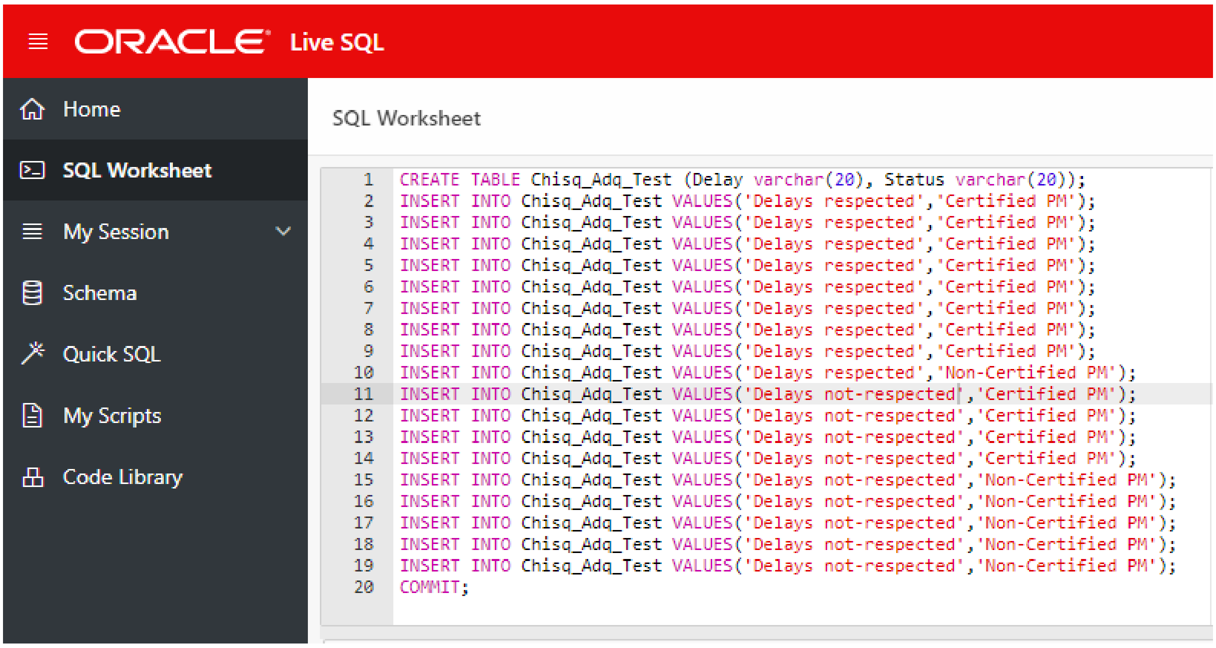


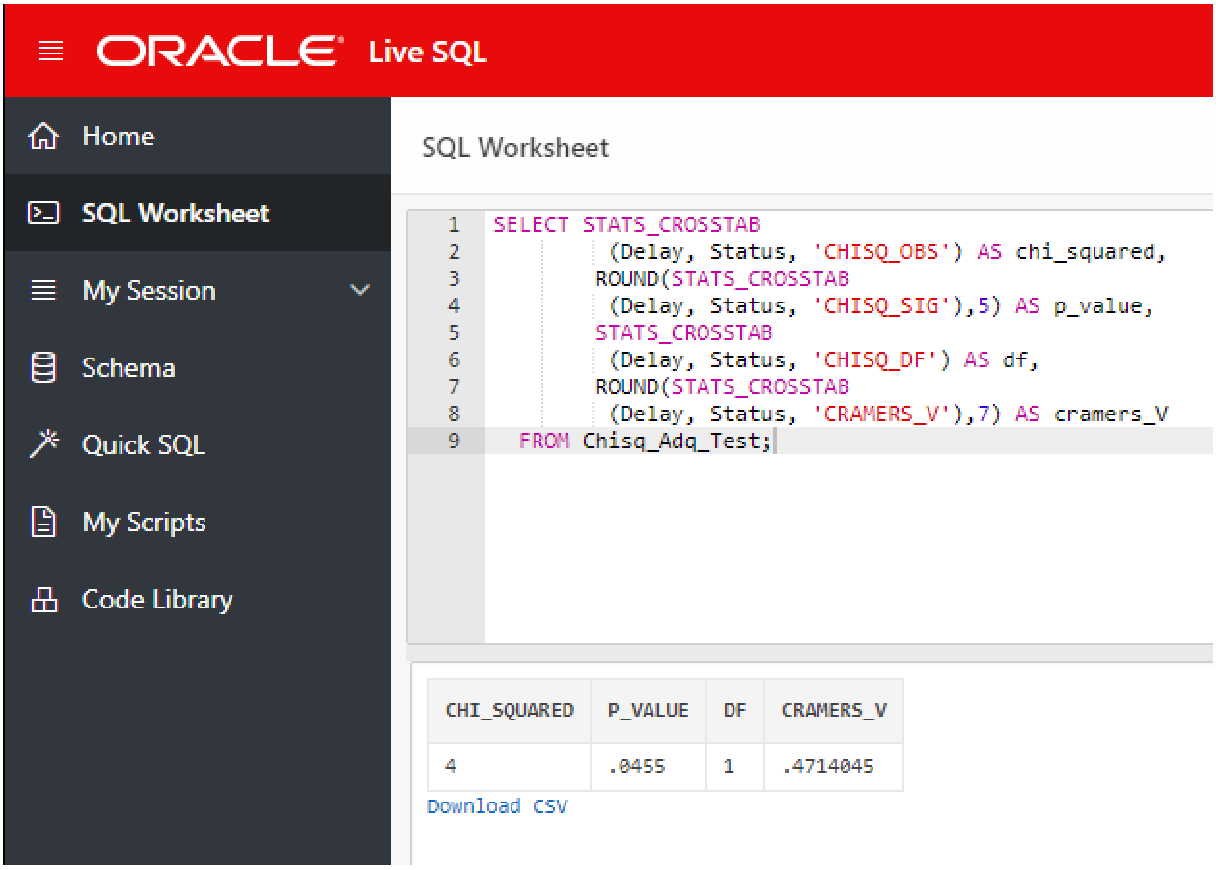


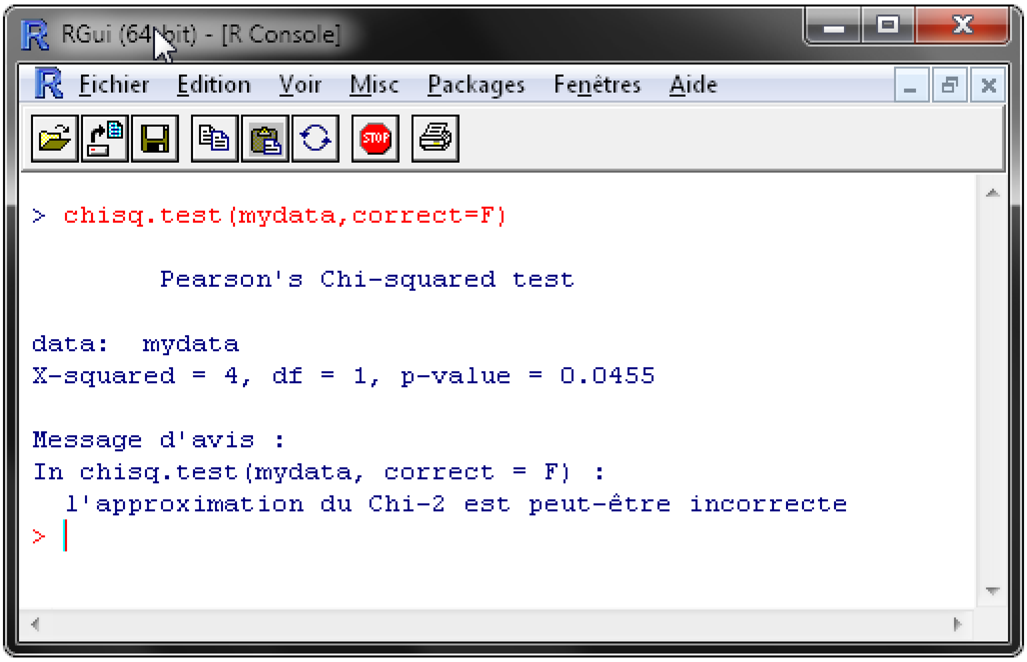


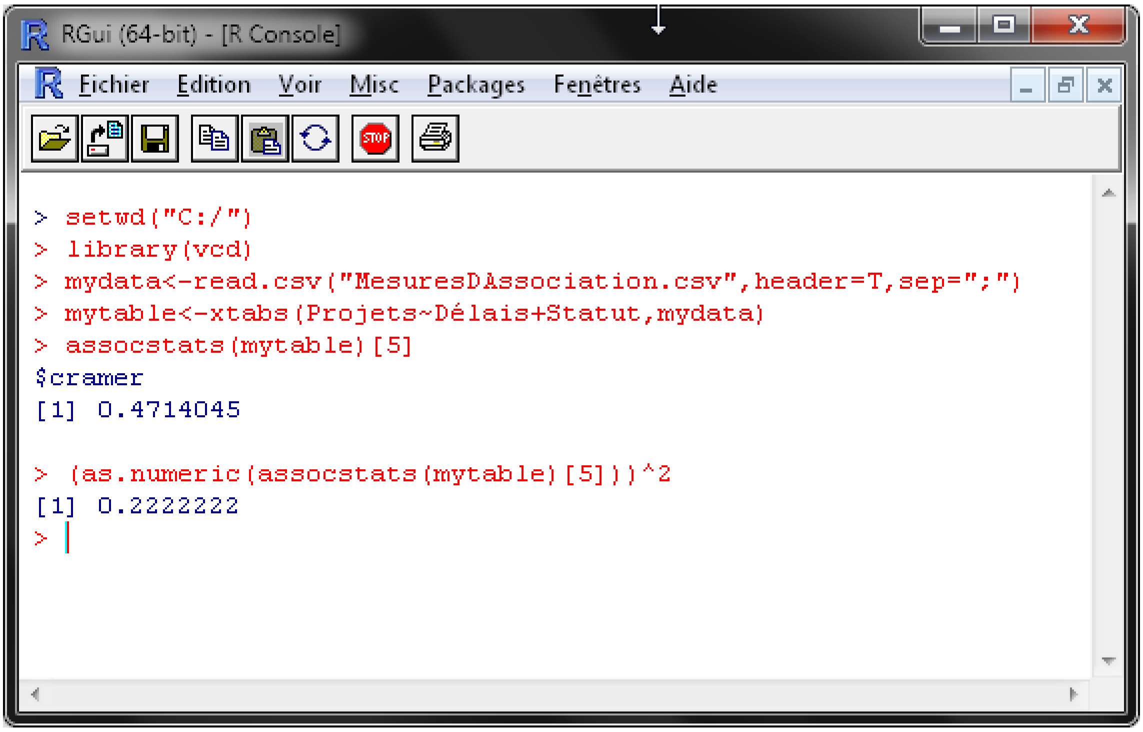
11.6 Chi-square adequation test with Yate's correction and Cramèrs' V



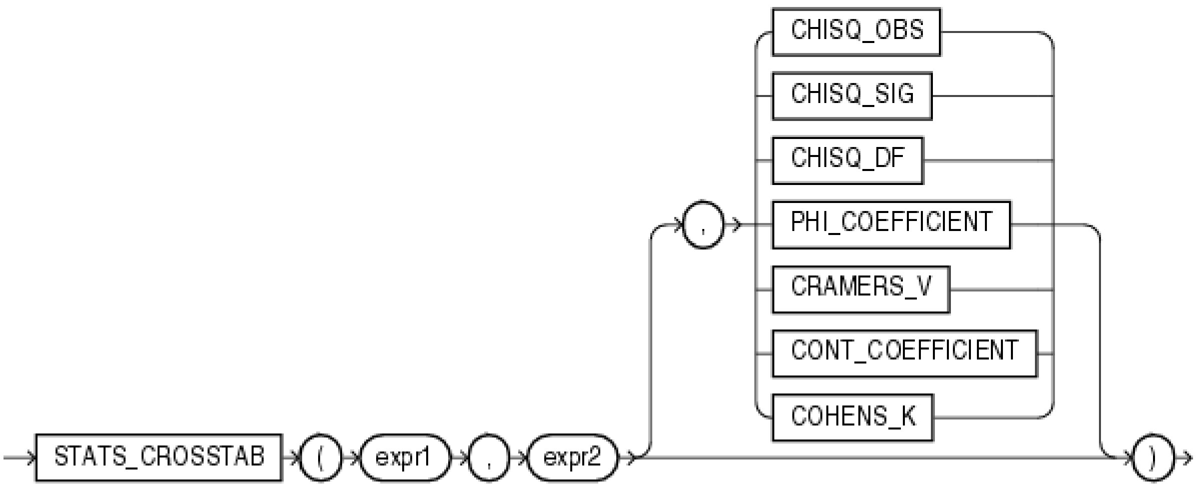


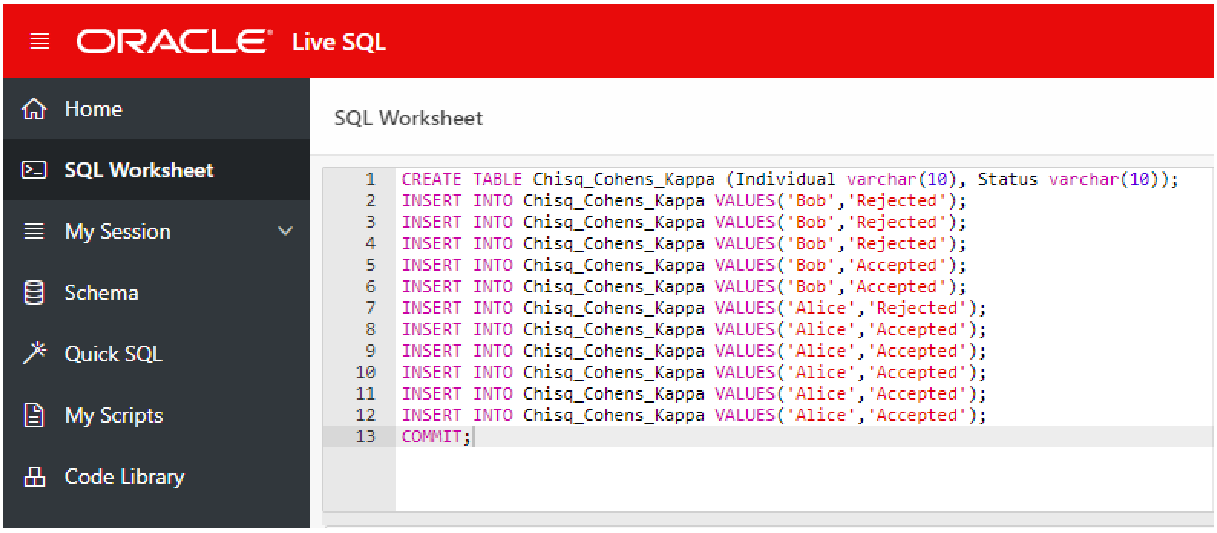

ORACLE Live SQL
Home
SQL Worksheet
My Session
Schema
Quick SQL
My Scripts
Code Library
SQL Worksheet
SELECT STATS_CROSSTAB
(Delay, Status, 'CHISQ_OBS') AS chi_squared,
ROUND(STATS_CROSSTAB
(Delay, Status, 'CHISQ_SIG'),5) AS p_value,
STATS_CROSSTAB
(Delay, Status, 'CHISQ_DF') AS df,
ROUND(STATS_CROSSTAB
(Delay, Status, 'CRAMERS_V'),7) AS cramers_V
FROM Chisq_Adq_Test;
CHI_SQUARED P_VALUE DF CRAMERS_V
4 .0455 1 .4714045
Download CSV

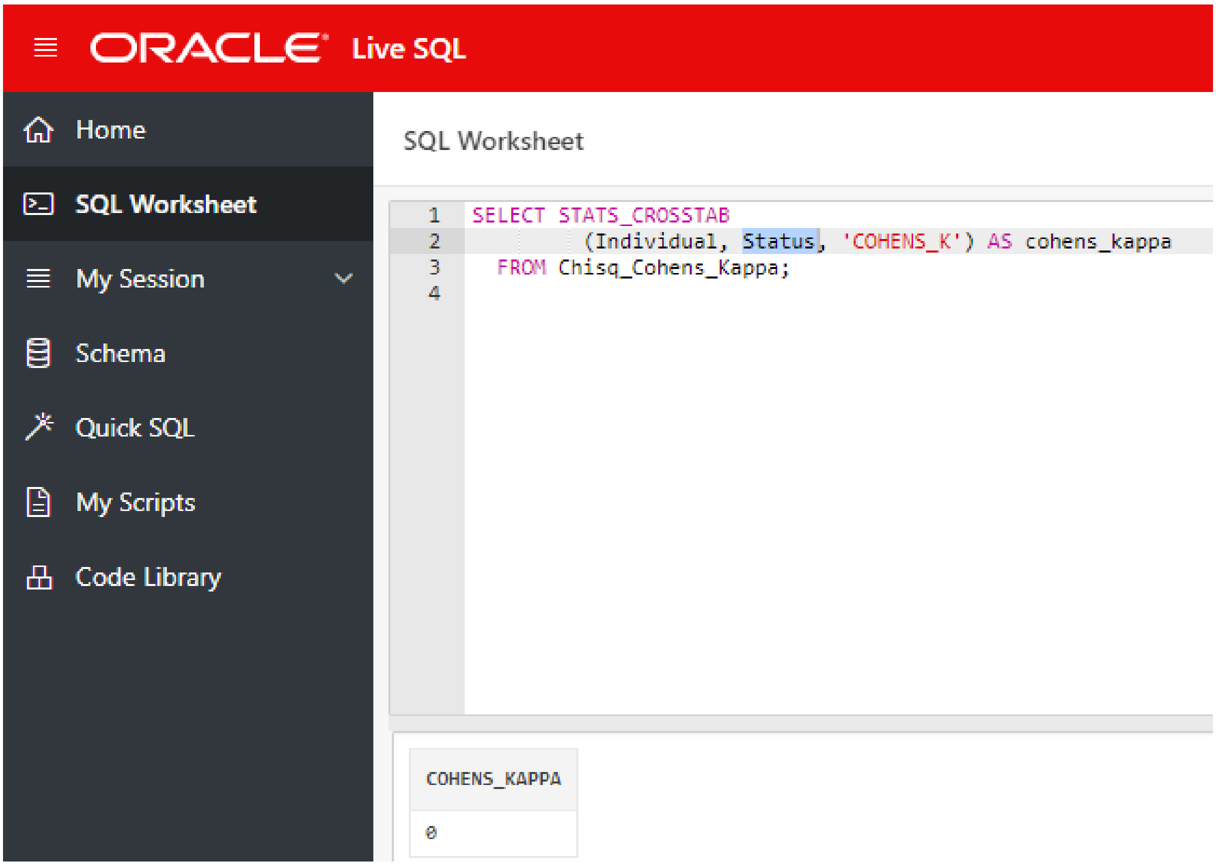

RGui (64-bit) - [R Console]
Fichier Edition Voir Misc Packages Fenêtres Aide
> chisq.test(mydata,correct=F)
Pearson's Chi-squared test
data: mydata
X-squared = 4, df = 1, p-value = 0.0455
Message d'avis :
In chisq.test(mydata, correct = F) :
l'approximation du Chi-2 est peut-être incorrecte
>


RGui (64-bit) - [R Console]
Fichier Edition Voir Misc Packages Fenêtres Aide
> setwd("C:/")
> library(vcd)
> mydata<-read.csv("MesuresDAssociation.csv",header=T,sep=";")
> mytable<-xtabs(Projets~Délais+Statut,mydata)
> assocstats(mytable)[5]
$cramer
[1] 0.4714045
> (as.numeric(assocstats(mytable)[5]))^2
[1] 0.2222222
>

11.7 Chi-square adequation test with Cohens kappa

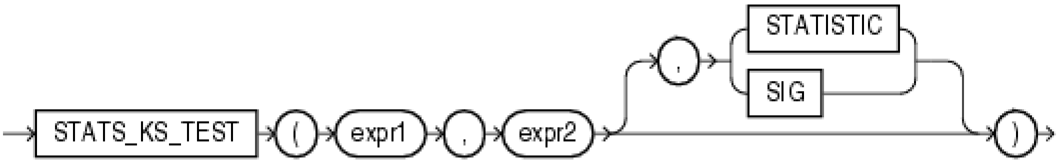


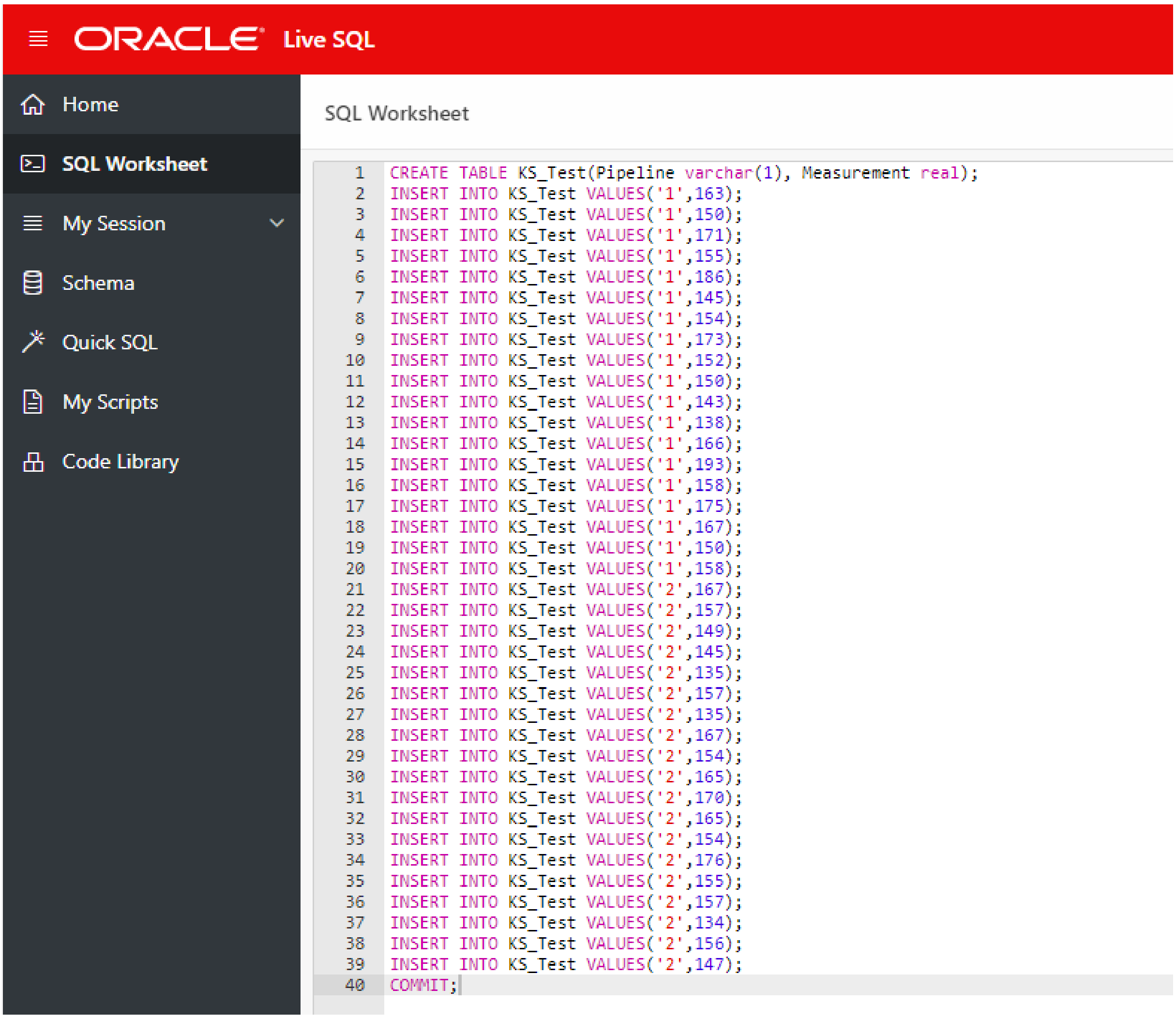




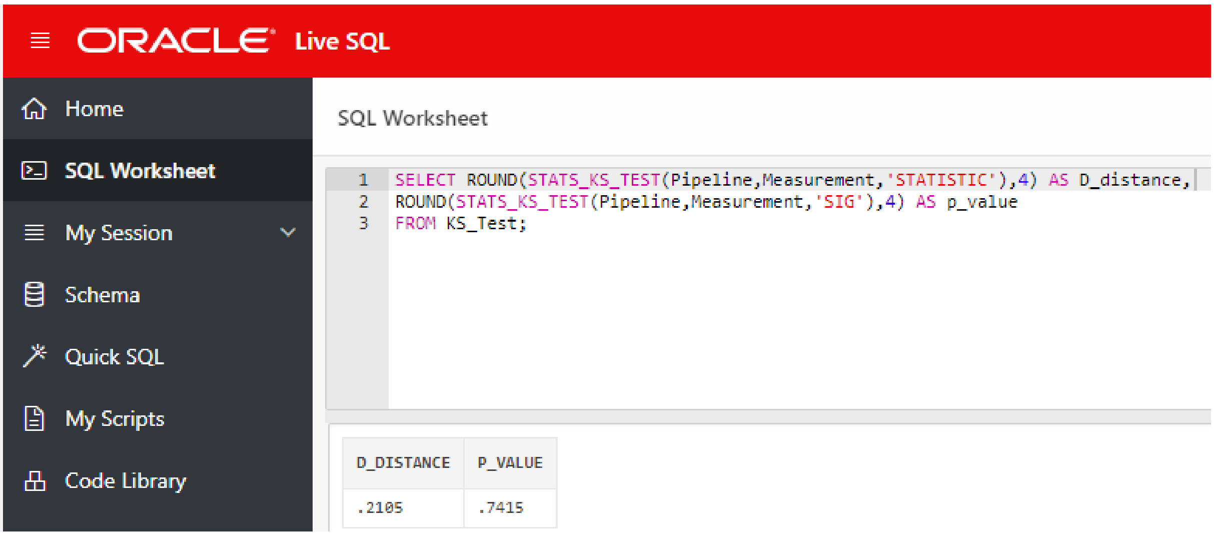
In fact, whatever the table used or the data, Oracle returns always 0 for Cohen's kappa. This mean that there is almost surely an issue or a bug with this parameter.

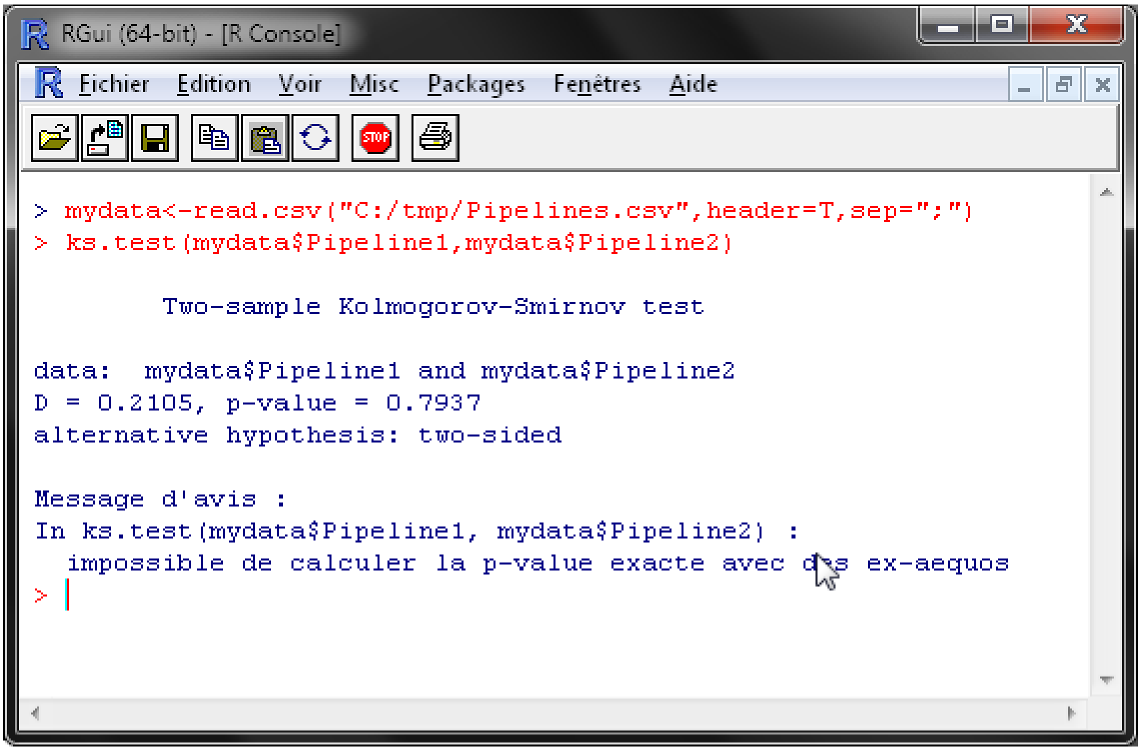
11.8 Two Sample Kolmogorov-Smirnov Adequation Test



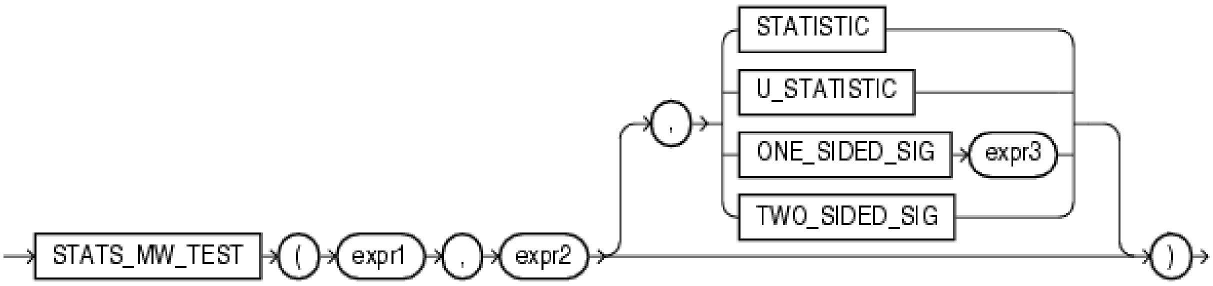


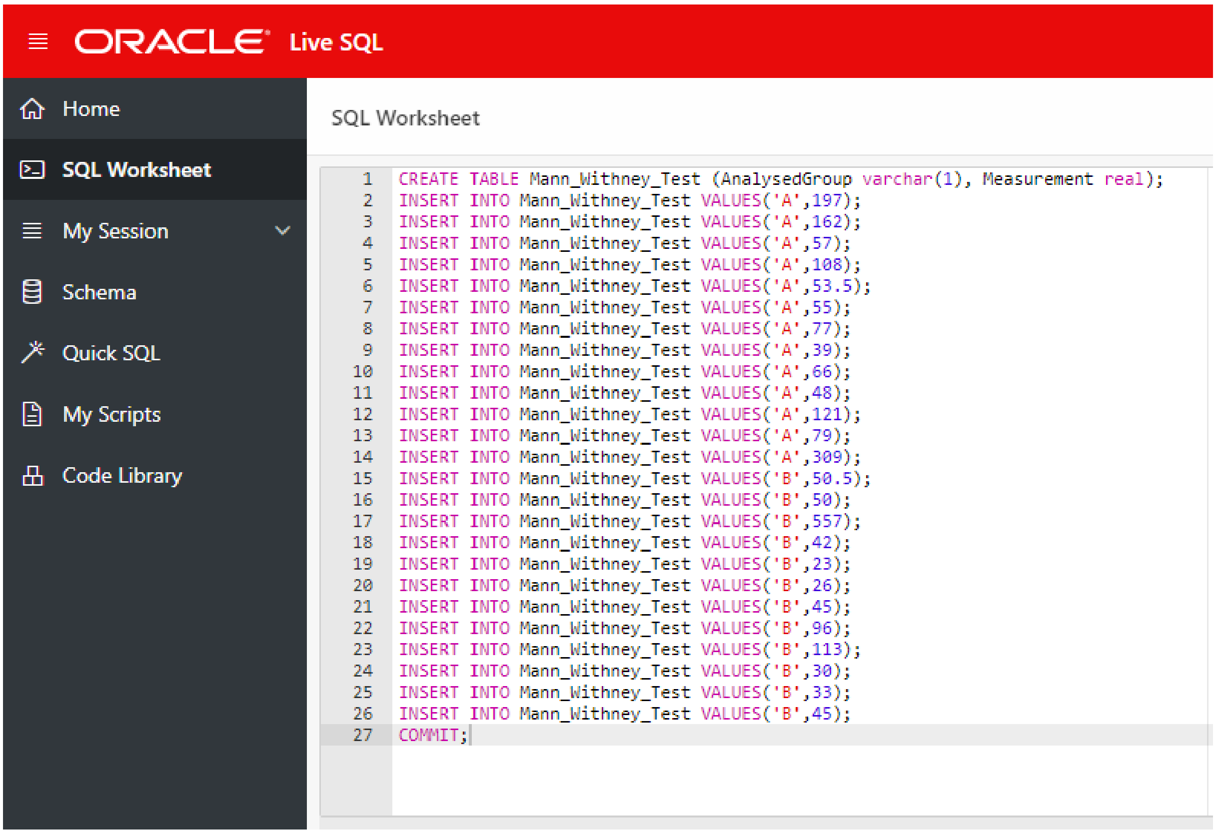

```sql
CREATE TABLE KS_Test(Pipeline varchar(1), Measurement real);
INSERT INTO KS_Test VALUES('1',163);
INSERT INTO KS_Test VALUES('1',150);
INSERT INTO KS_Test VALUES('1',171);
INSERT INTO KS_Test VALUES('1',155);
INSERT INTO KS_Test VALUES('1',186);
INSERT INTO KS_Test VALUES('1',145);
INSERT INTO KS_Test VALUES('1',154);
INSERT INTO KS_Test VALUES('1',173);
INSERT INTO KS_Test VALUES('1',152);
INSERT INTO KS_Test VALUES('1',150);
INSERT INTO KS_Test VALUES('1',143);
INSERT INTO KS_Test VALUES('1',138);
INSERT INTO KS_Test VALUES('1',166);
INSERT INTO KS_Test VALUES('1',193);
INSERT INTO KS_Test VALUES('1',158);
INSERT INTO KS_Test VALUES('1',175);
INSERT INTO KS_Test VALUES('1',167);
INSERT INTO KS_Test VALUES('1',150);
INSERT INTO KS_Test VALUES('1',158);
INSERT INTO KS_Test VALUES('2',167);
INSERT INTO KS_Test VALUES('2',157);
INSERT INTO KS_Test VALUES('2',149);
INSERT INTO KS_Test VALUES('2',145);
INSERT INTO KS_Test VALUES('2',135);
INSERT INTO KS_Test VALUES('2',157);
INSERT INTO KS_Test VALUES('2',135);
INSERT INTO KS_Test VALUES('2',167);
INSERT INTO KS_Test VALUES('2',154);
INSERT INTO KS_Test VALUES('2',165);
INSERT INTO KS_Test VALUES('2',170);
INSERT INTO KS_Test VALUES('2',165);
INSERT INTO KS_Test VALUES('2',154);
INSERT INTO KS_Test VALUES('2',176);
INSERT INTO KS_Test VALUES('2',155);
INSERT INTO KS_Test VALUES('2',157);
INSERT INTO KS_Test VALUES('2',134);
INSERT INTO KS_Test VALUES('2',156);
INSERT INTO KS_Test VALUES('2',147);
COMMIT;
```

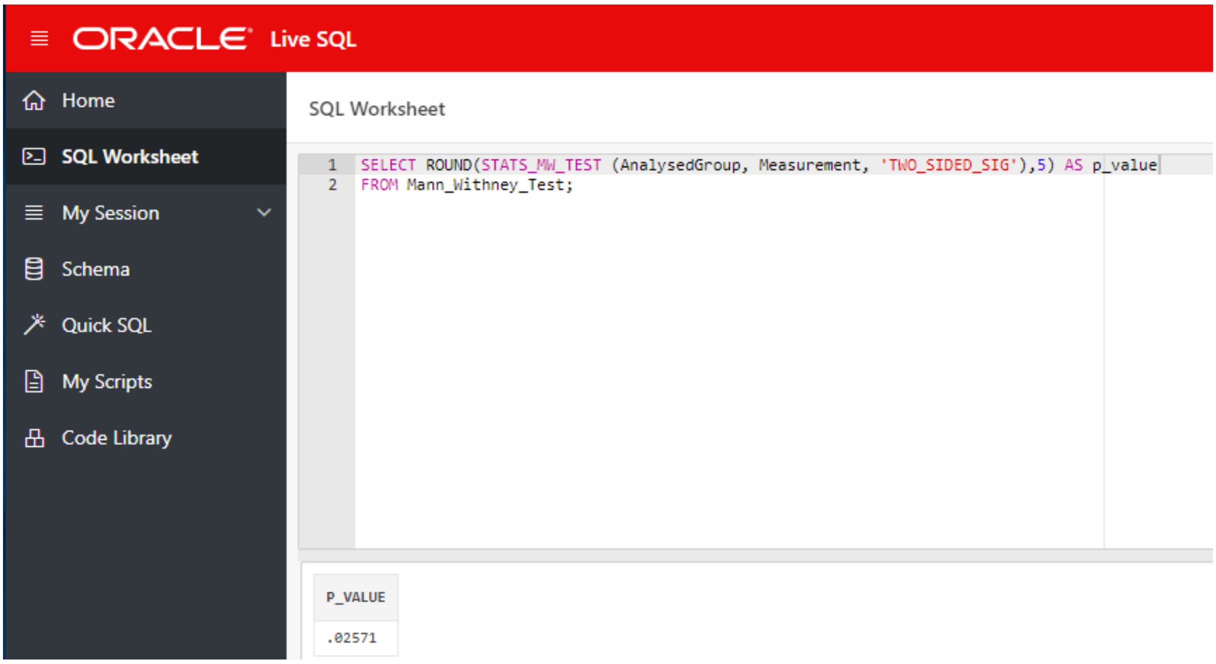

ORACLE Live SQL
Home
SQL Worksheet
My Session
Schema
Quick SQL
My Scripts
Code Library
SQL Worksheet
SELECT ROUND(STATS_KS_TEST(Pipeline,Measurement,'STATISTIC'),4) AS D_distance,
ROUND(STATS_KS_TEST(Pipeline,Measurement,'SIG'),4) AS p_value
FROM KS_Test;
D_DISTANCE P_VALUE
.2105 .7415

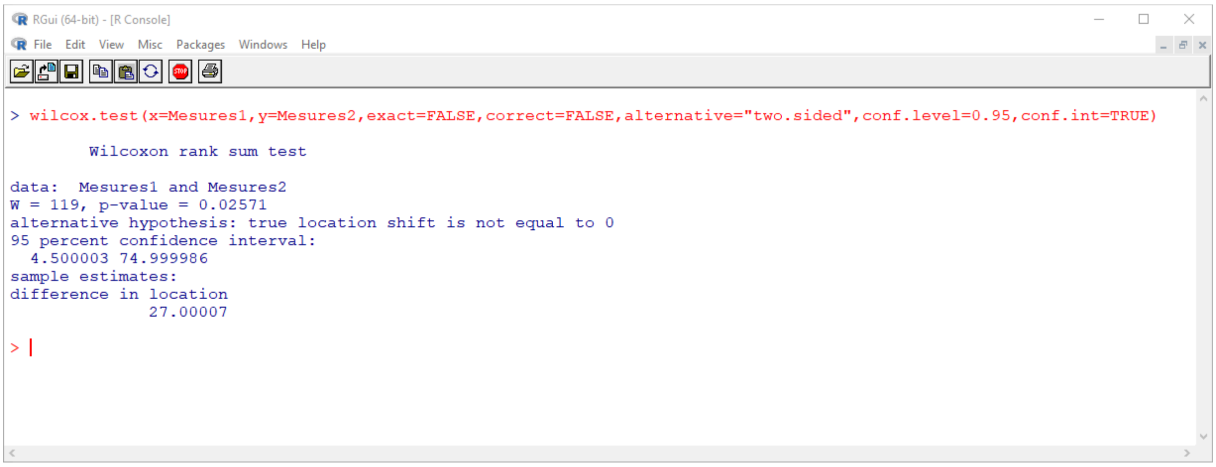

RGui (64-bit) - [R Console]
Fichier Edition Voir Misc Packages Fenêtres Aide
> mydata<-read.csv("C:/tmp/Pipelines.csv",header=T,sep=";")
> ks.test(mydata$Pipeline1,mydata$Pipeline2)
Two-sample Kolmogorov-Smirnov test
data: mydata$Pipeline1 and mydata$Pipeline2
D = 0.2105, p-value = 0.7937
alternative hypothesis: two-sided
Message d'avis :
In ks.test(mydata$Pipeline1, mydata$Pipeline2) :
impossible de calculer la p-value exacte avec des ex-aequos

11.9 Mann-Withney (Wilcoxon Rank) Test

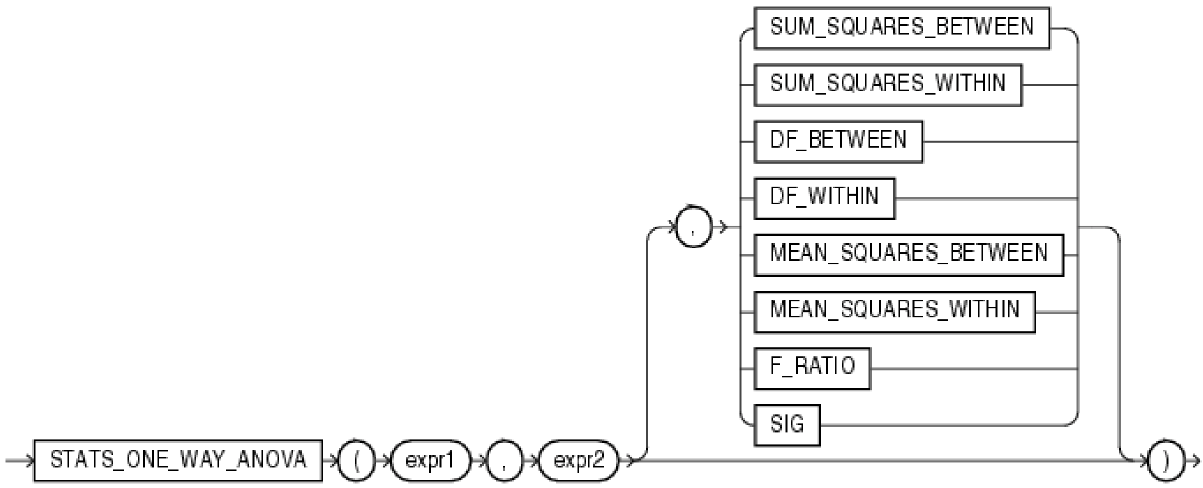


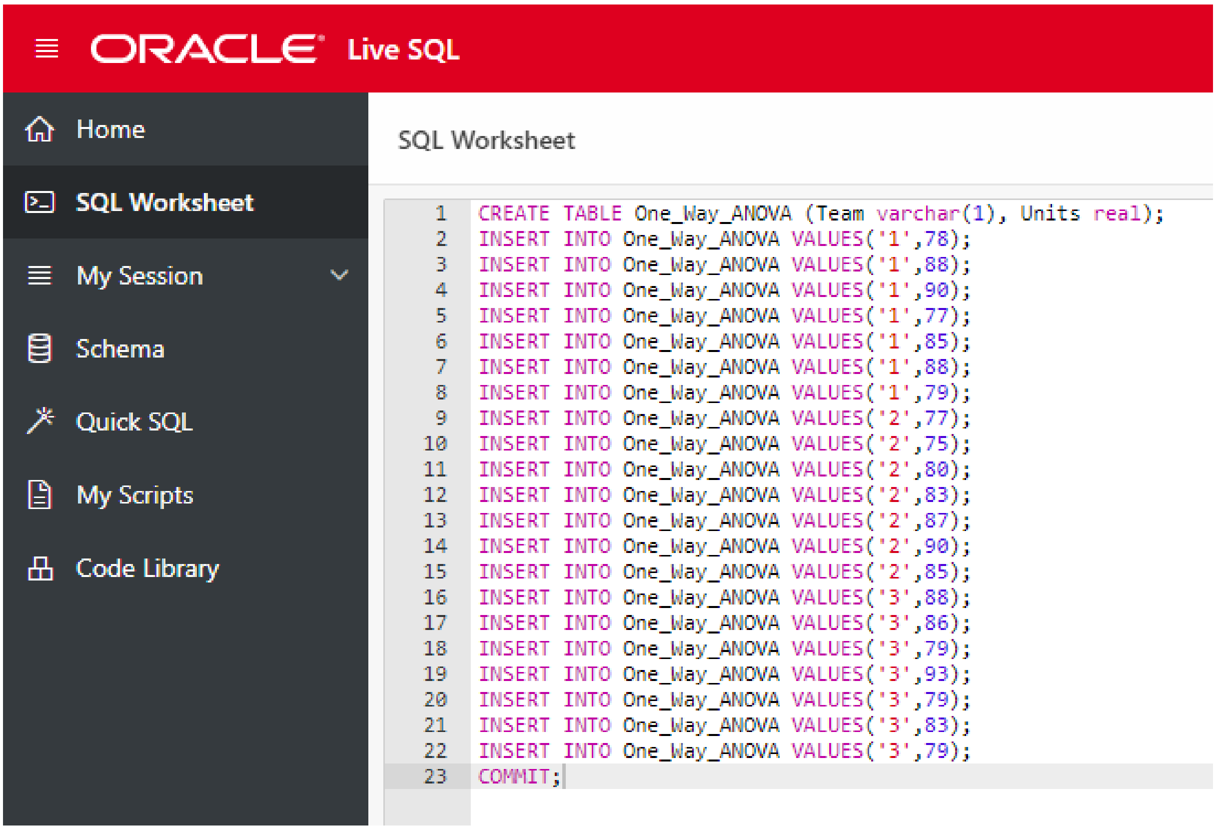


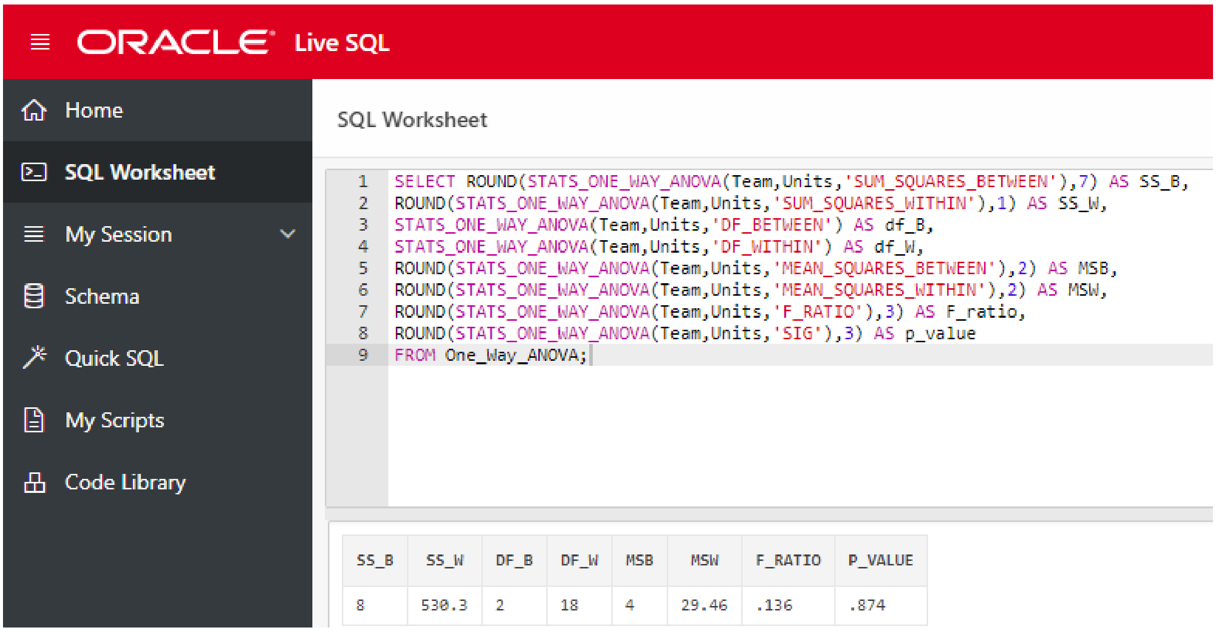

ORACLE Live SQL
Home
SQL Worksheet
My Session
Schema
Quick SQL
My Scripts
Code Library
SQL Worksheet
1 SELECT ROUND(STATS_MW_TEST (AnalysedGroup, Measurement, 'TWO_SIDED_SIG'),5) AS p_value
2 FROM Mann_Withney_Test;
P_VALUE
.02571

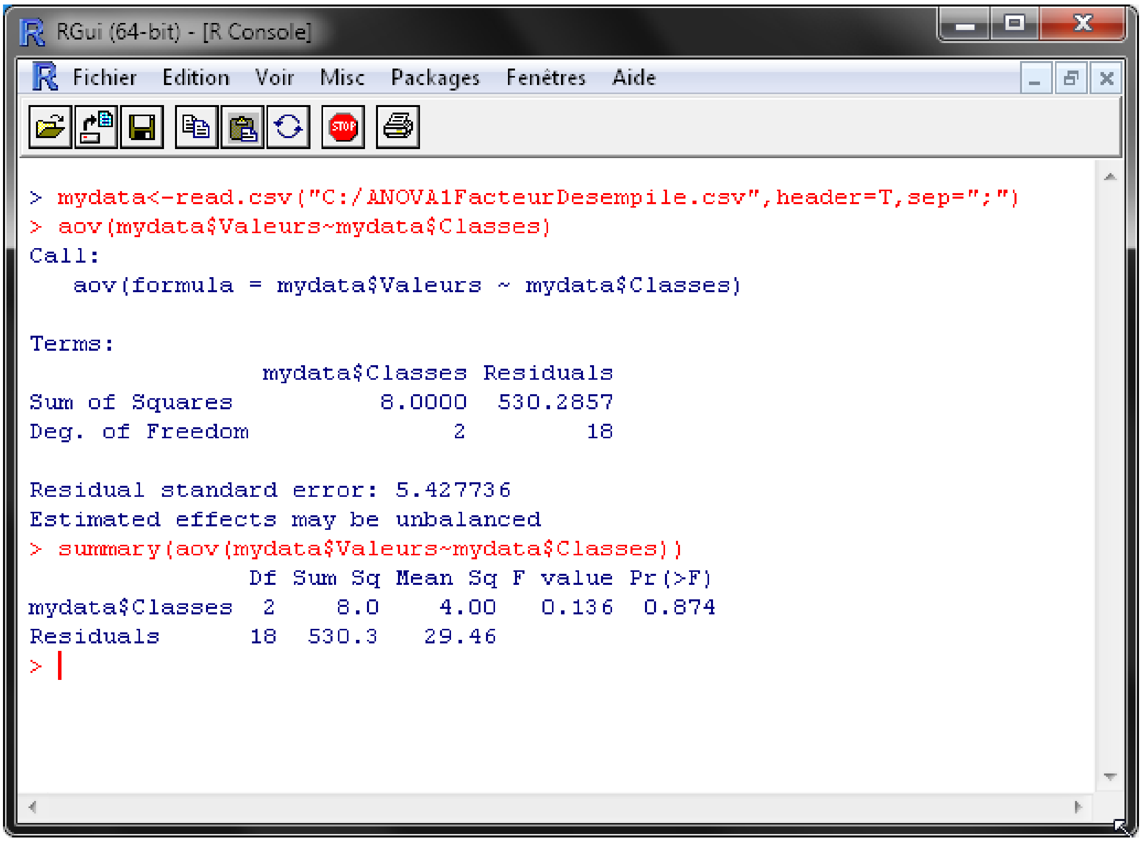

RGui (64-bit) - [R Console]
File Edit View Misc Packages Windows Help
> wilcox.test(x=Mesures1,y=Mesures2,exact=FALSE,correct=FALSE,alternative="two.sided",conf.level=0.95,conf.int=TRUE)
Wilcoxon rank sum test
data: Mesures1 and Mesures2
W = 119, p-value = 0.02571
alternative hypothesis: true location shift is not equal to 0
95 percent confidence interval:
4.500003 74.999986
sample estimates:
difference in location
27.00007
>

11.10 One-Way ANOVA

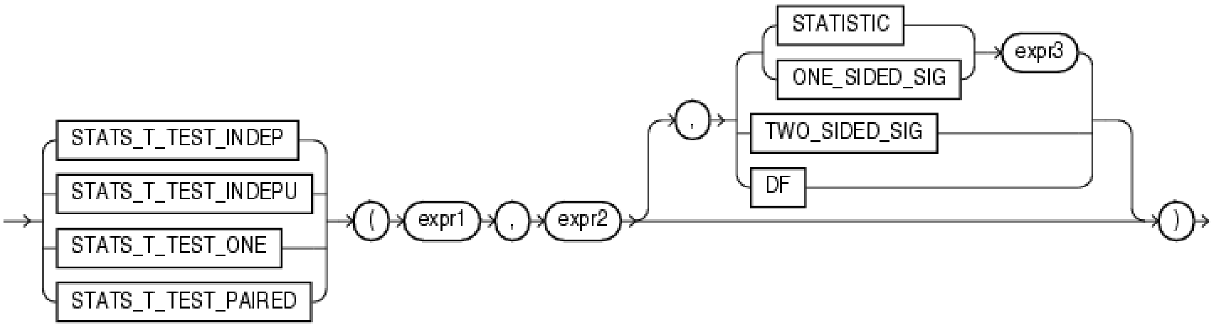




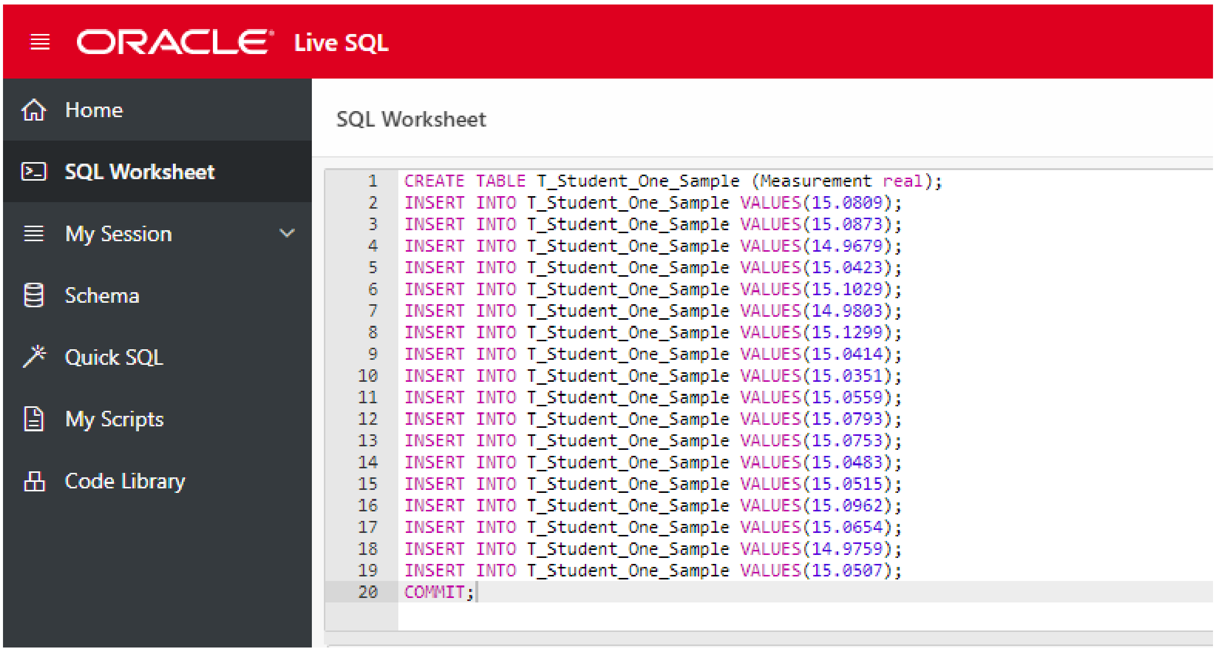

ORACLE Live SQL
Home
SQL Worksheet
My Session
Schema
Quick SQL
My Scripts
Code Library
SQL Worksheet
SELECT ROUND(STATS_ONE_WAY_ANOVA(Team,Units,'SUM_SQUARES_BETWEEN'),7) AS SS_B,
ROUND(STATS_ONE_WAY_ANOVA(Team,Units,'SUM_SQUARES_WITHIN'),1) AS SS_W,
STATS_ONE_WAY_ANOVA(Team,Units,'DF_BETWEEN') AS df_B,
STATS_ONE_WAY_ANOVA(Team,Units,'DF_WITHIN') AS df_W,
ROUND(STATS_ONE_WAY_ANOVA(Team,Units,'MEAN_SQUARES_BETWEEN'),2) AS MSB,
ROUND(STATS_ONE_WAY_ANOVA(Team,Units,'MEAN_SQUARES_WITHIN'),2) AS MSW,
ROUND(STATS_ONE_WAY_ANOVA(Team,Units,'F_RATIO'),3) AS F_ratio,
ROUND(STATS_ONE_WAY_ANOVA(Team,Units,'SIG'),3) AS p_value
FROM One_Way_ANOVA;
SS_B SS_W DF_B DF_W MSB MSW F_RATIO P_VALUE
8 530.3 2 18 4 29.46 .136 .874

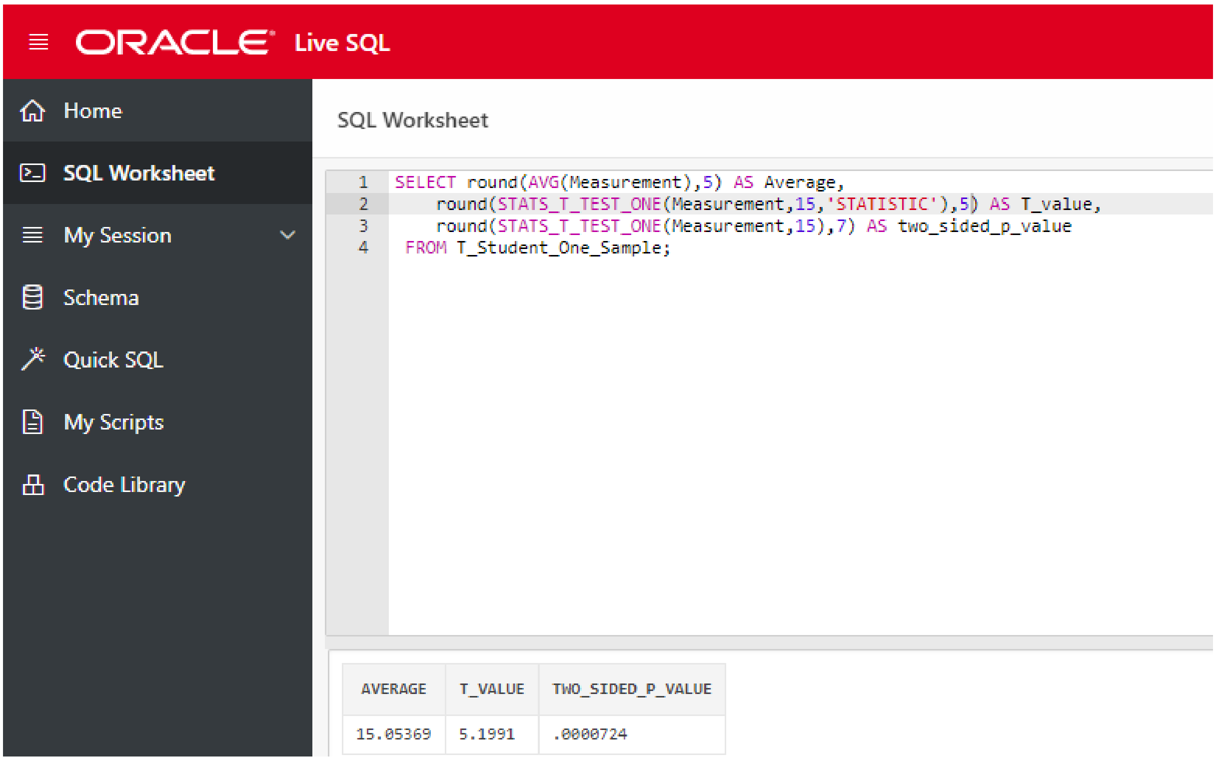

RGui (64-bit) - [R Console]
Fichier Edition Voir Misc Packages Fenêtres Aide
> mydata<-read.csv("C:/ANOVA1FacteurDesempile.csv",header=T,sep=";")
> aov(mydata$Valeurs~mydata$Classes)
Call:
aov(formula = mydata$Valeurs ~ mydata$Classes)
Terms:
mydata$Classes Residuals
Sum of Squares 8.0000 530.2857
Deg. of Freedom 2 18
Residual standard error: 5.427736
Estimated effects may be unbalanced
> summary(aov(mydata$Valeurs~mydata$Classes))
Df Sum Sq Mean Sq F value Pr(>F)
mydata$Classes 2 8.0 4.00 0.136 0.874
Residuals 18 530.3 29.46
>

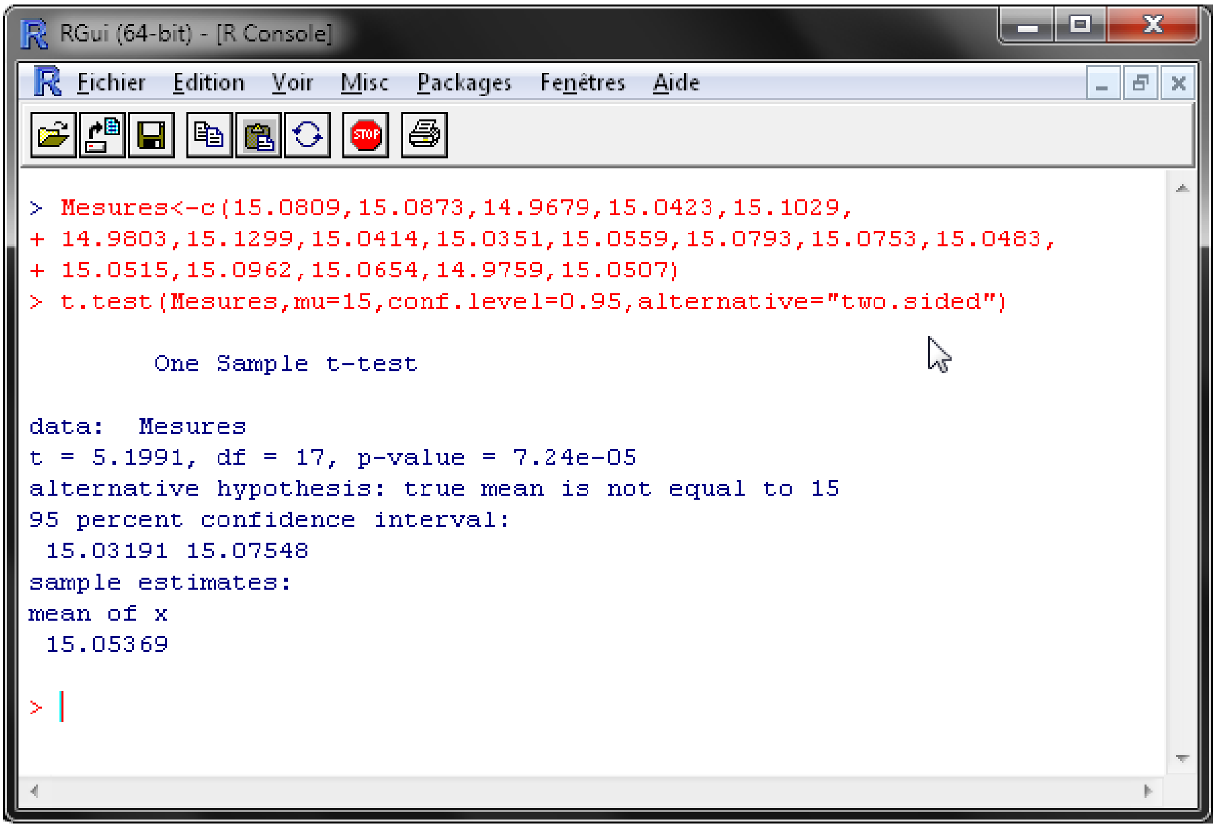
11.11 Student-T test



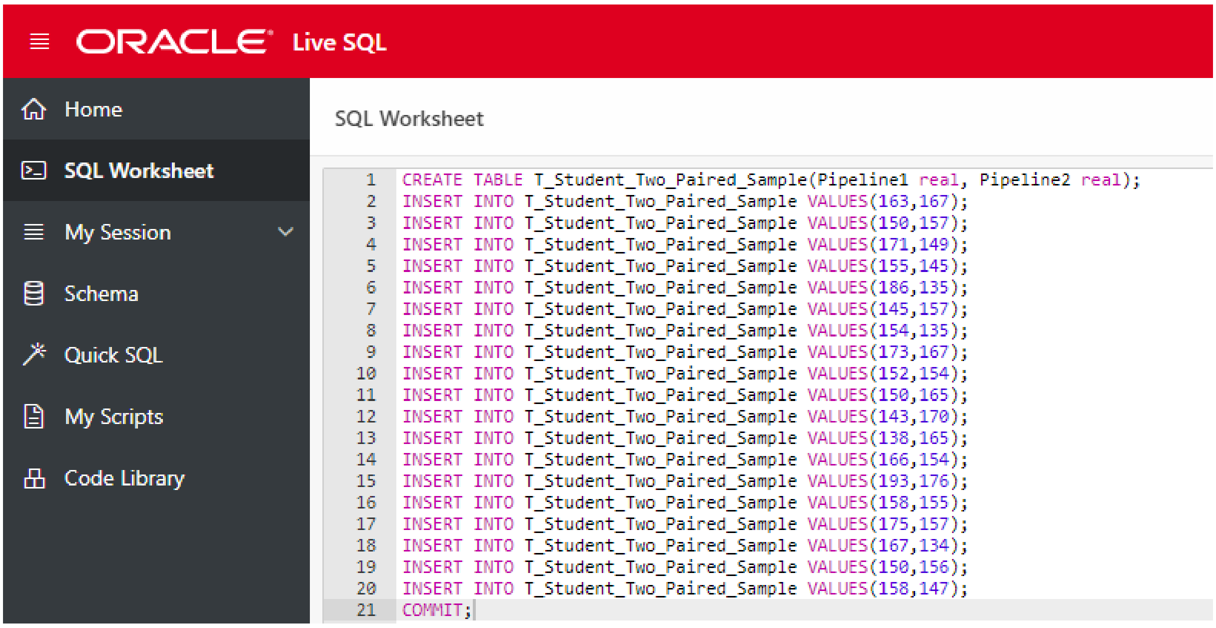
11.11.1 One sample T-test

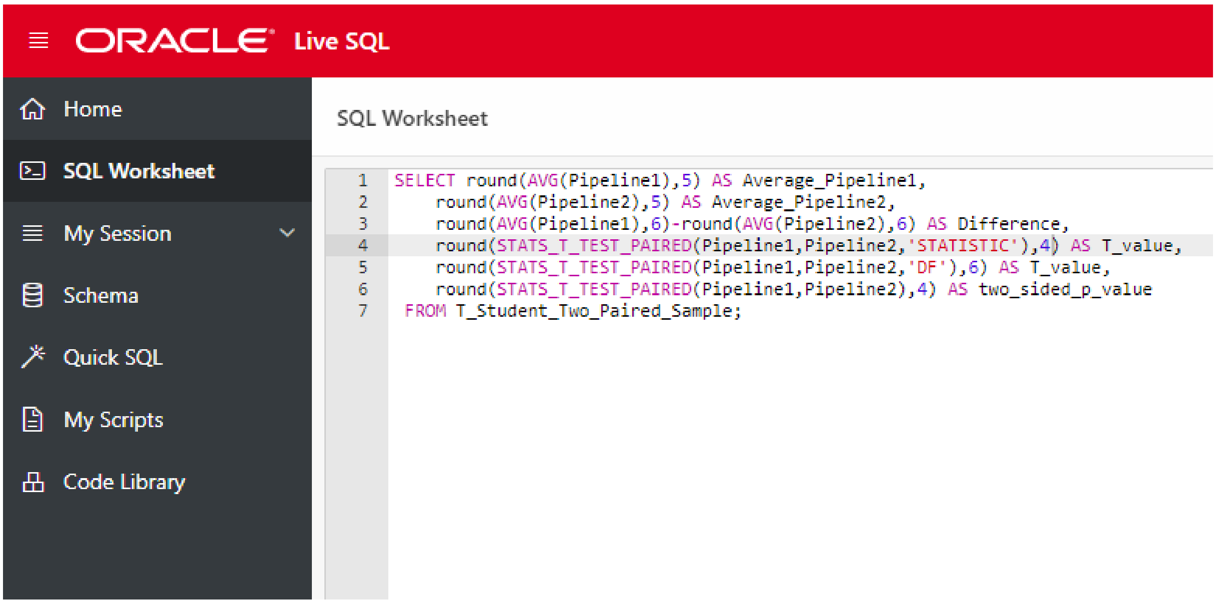


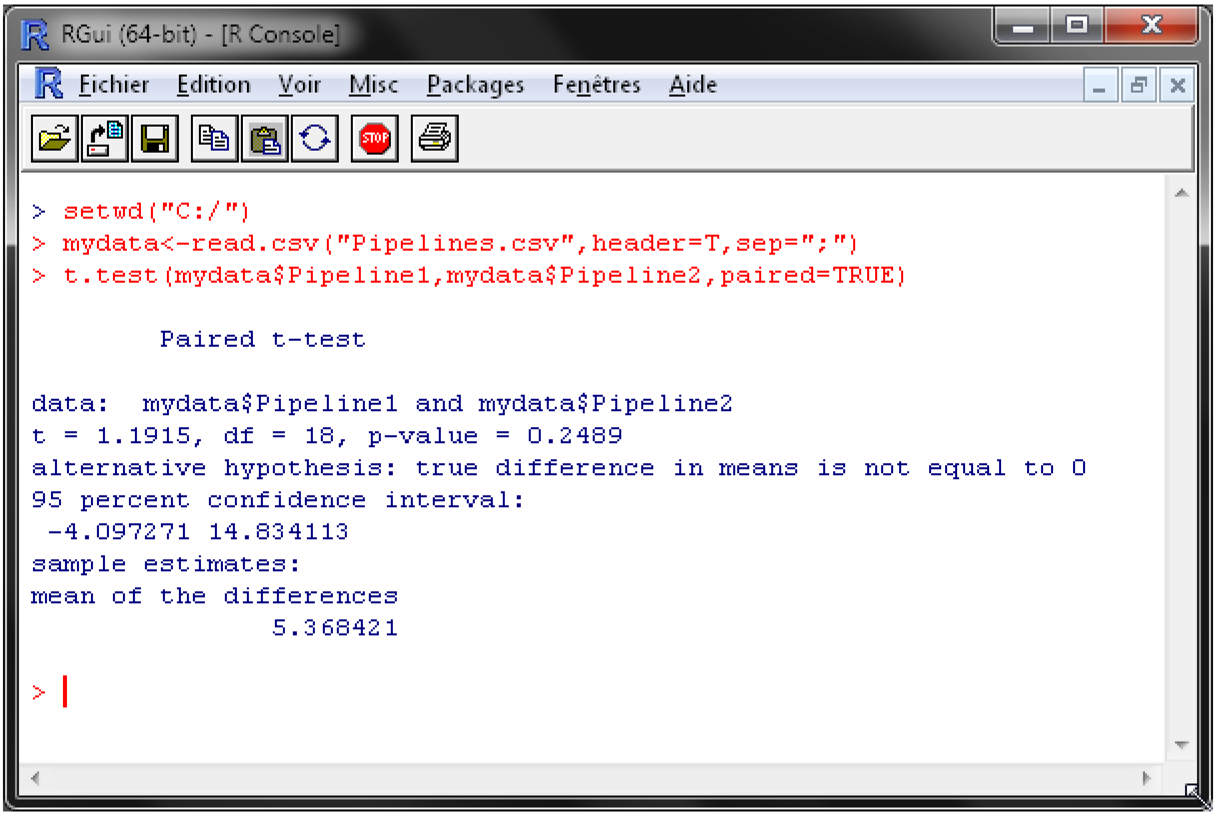

ORACLE Live SQL
Home
SQL Worksheet
My Session
Schema
Quick SQL
My Scripts
Code Library
SQL Worksheet
1 SELECT round(AVG(Measurement),5) AS Average,
2 round(STATS_T_TEST_ONE(Measurement,15,'STATISTIC'),5) AS T_value,
3 round(STATS_T_TEST_ONE(Measurement,15),7) AS two_sided_p_value
4 FROM T_Student_One_Sample;
AVERAGE T_VALUE TWO_SIDED_P_VALUE
15.05369 5.1991 .0000724


RGui (64-bit) - [R Console]
Fichier Edition Voir Misc Packages Fenêtres Aide
> Mesures<-c(15.0809,15.0873,14.9679,15.0423,15.1029,
+ 14.9803,15.1299,15.0414,15.0351,15.0559,15.0793,15.0753,15.0483,
+ 15.0515,15.0962,15.0654,14.9759,15.0507)
> t.test(Mesures,mu=15,conf.level=0.95,alternative="two.sided")
One Sample t-test
data: Mesures
t = 5.1991, df = 17, p-value = 7.24e-05
alternative hypothesis: true mean is not equal to 15
95 percent confidence interval:
15.03191 15.07548
sample estimates:
mean of x
15.05369
>

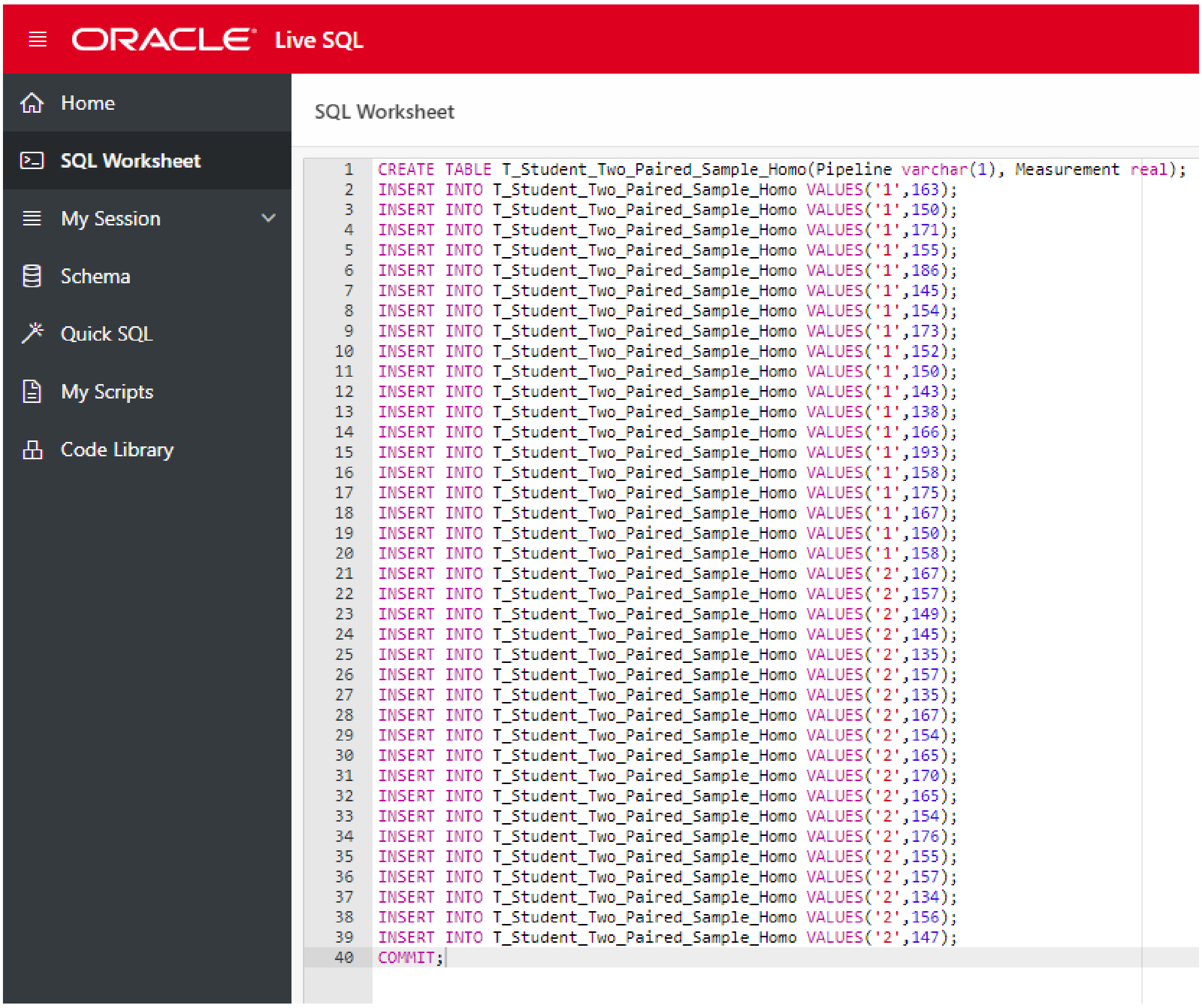
11.11.2 Two sample paired T-test

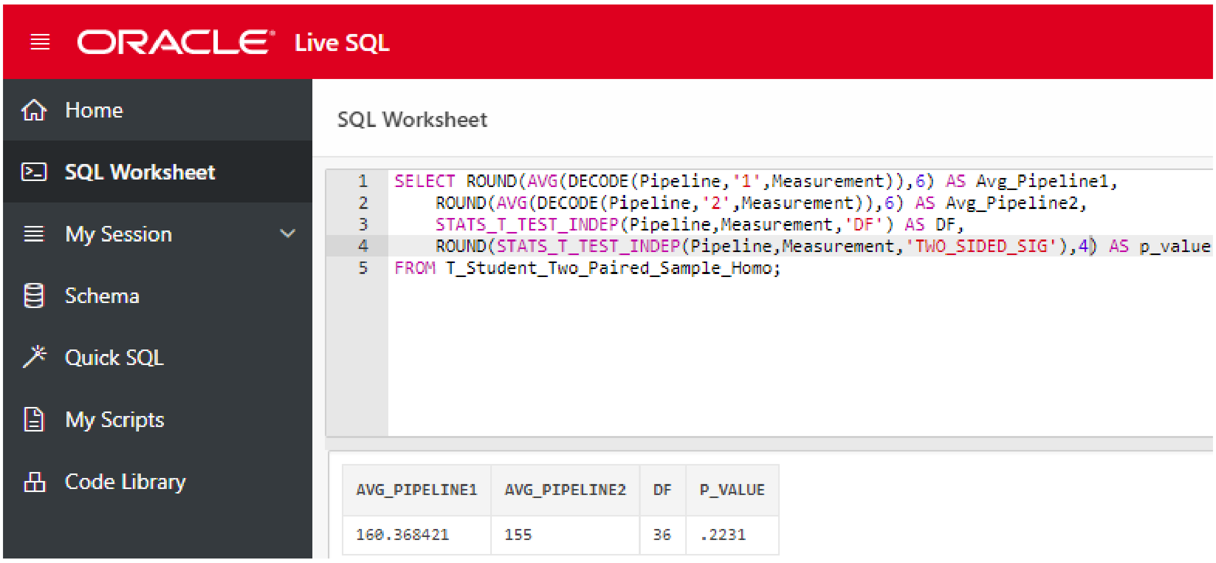


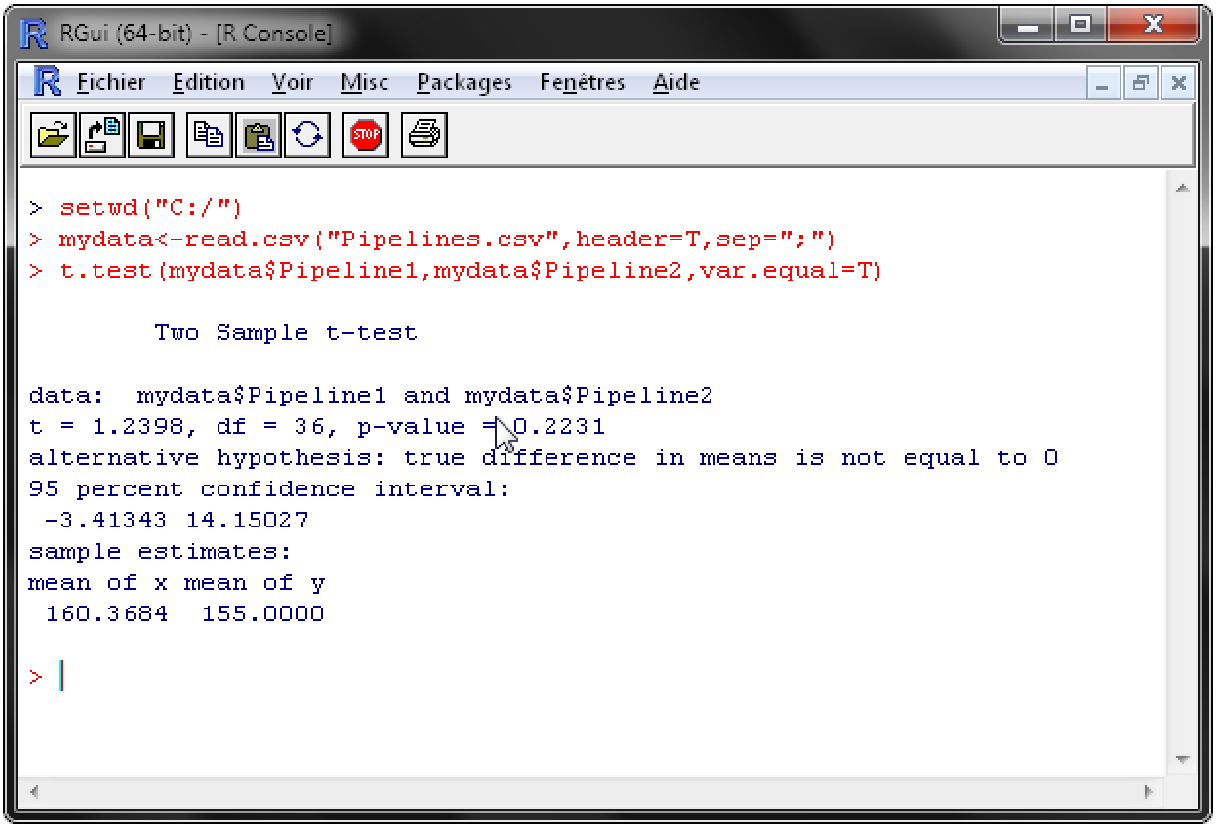



RGui (64-bit) - [R Console]
Fichier Edition Voir Misc Packages Fenêtres Aide
> setwd("C:/")
> mydata<-read.csv("Pipelines.csv",header=T,sep=";")
> t.test(mydata$Pipeline1,mydata$Pipeline2,paired=TRUE)
Paired t-test
data: mydata$Pipeline1 and mydata$Pipeline2
t = 1.1915, df = 18, p-value = 0.2489
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-4.097271 14.834113
sample estimates:
mean of the differences
5.368421
>

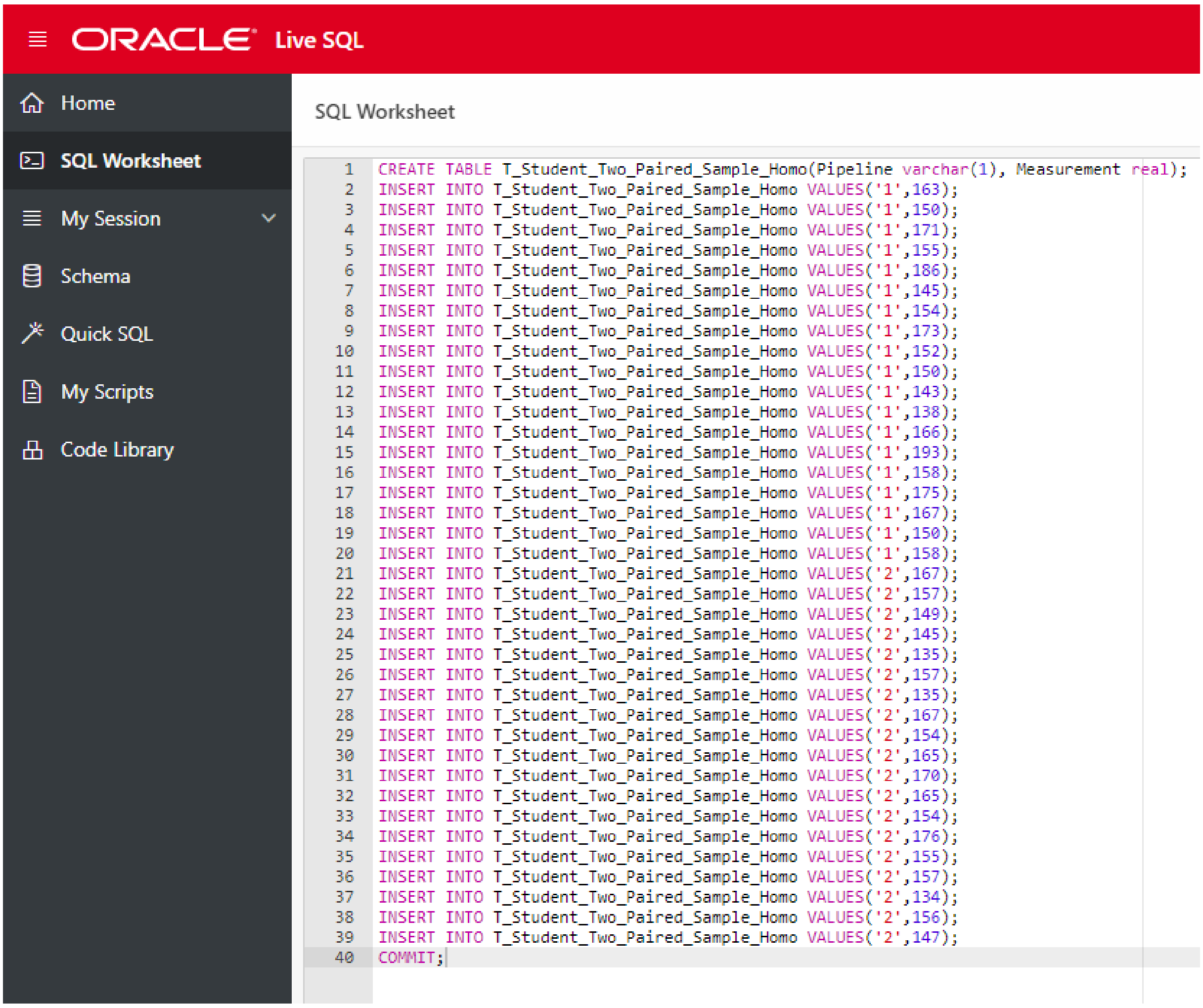
11.11.3 Two sample homoscedastic T-test

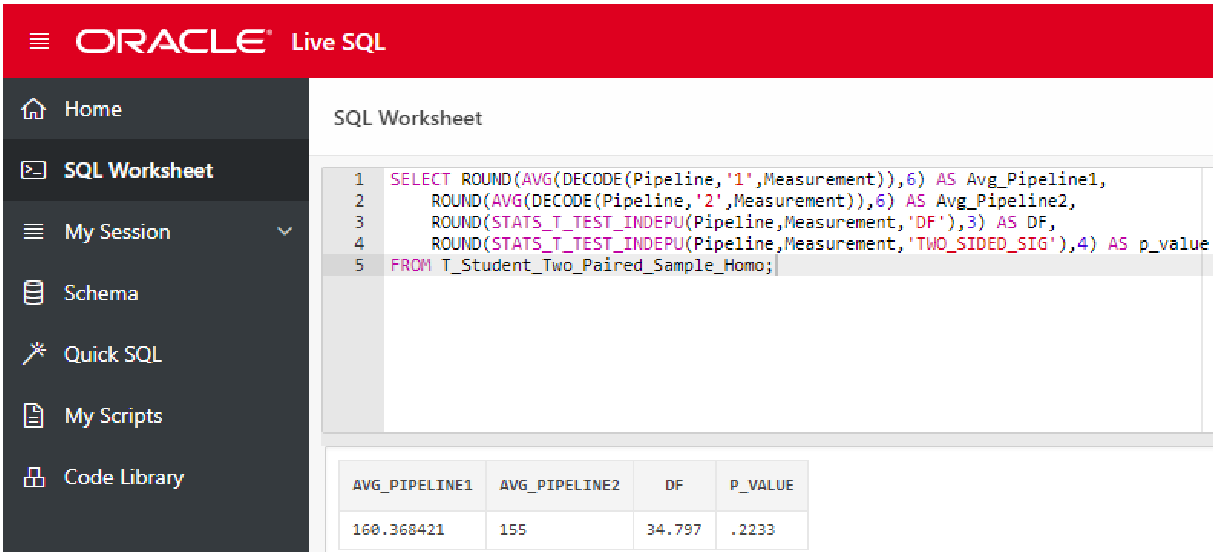


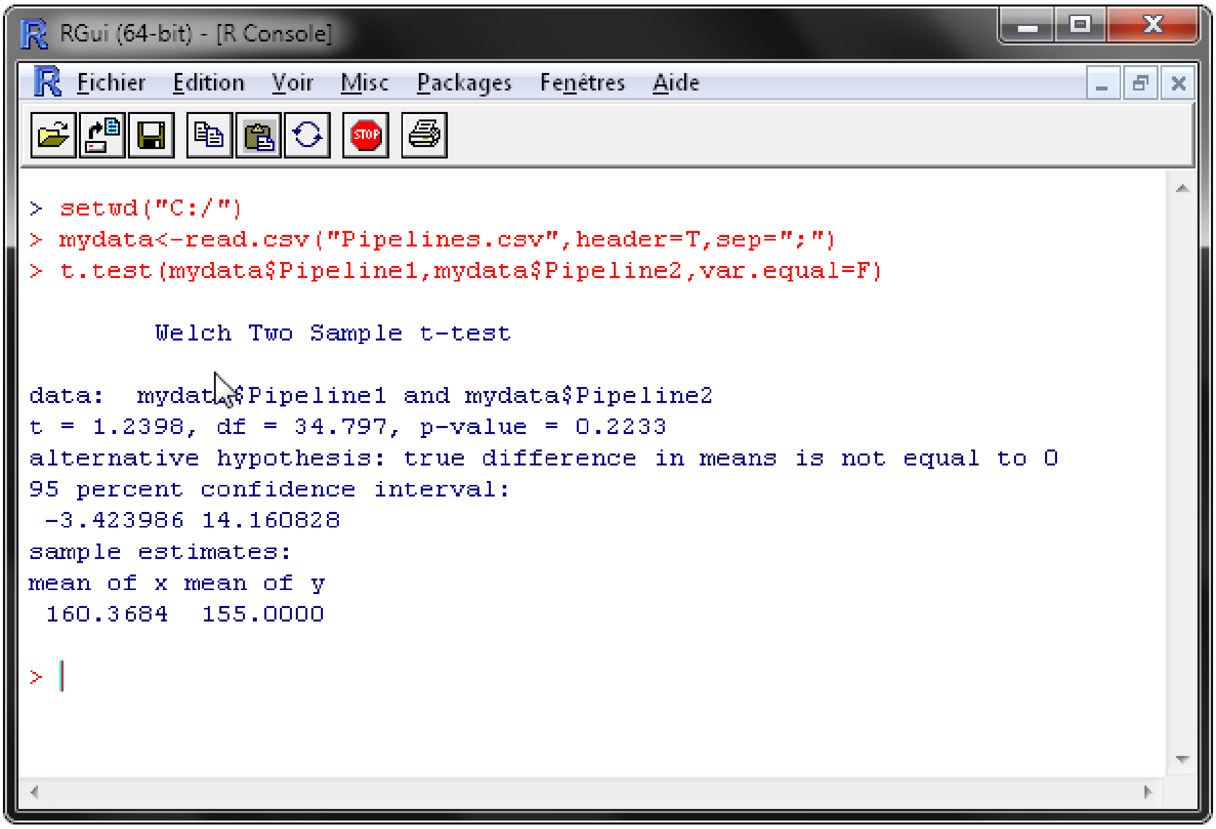



RGui (64-bit) - [R Console]
Fichier Edition Voir Misc Packages Fenêtres Aide
> setwd("C:/")
> mydata<-read.csv("Pipelines.csv",header=T,sep=";")
> t.test(mydata$Pipeline1,mydata$Pipeline2,var.equal=T)
Two Sample t-test
data: mydata$Pipeline1 and mydata$Pipeline2
t = 1.2398, df = 36, p-value = 0.2231
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-3.41343 14.15027
sample estimates:
mean of x mean of y
160.3684 155.0000
>

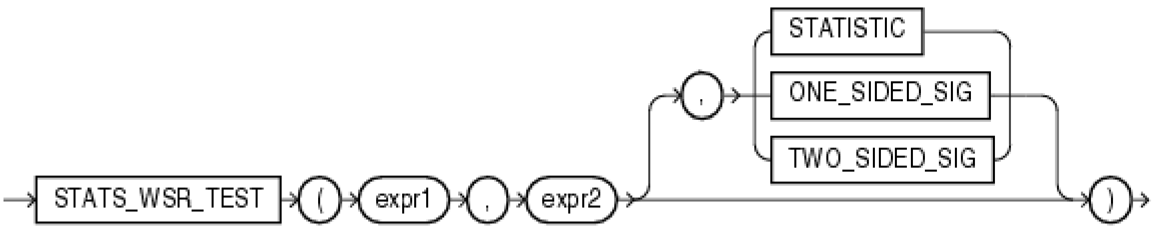
11.11.4 Two sample heteroscedastic T-test (Welch Test)

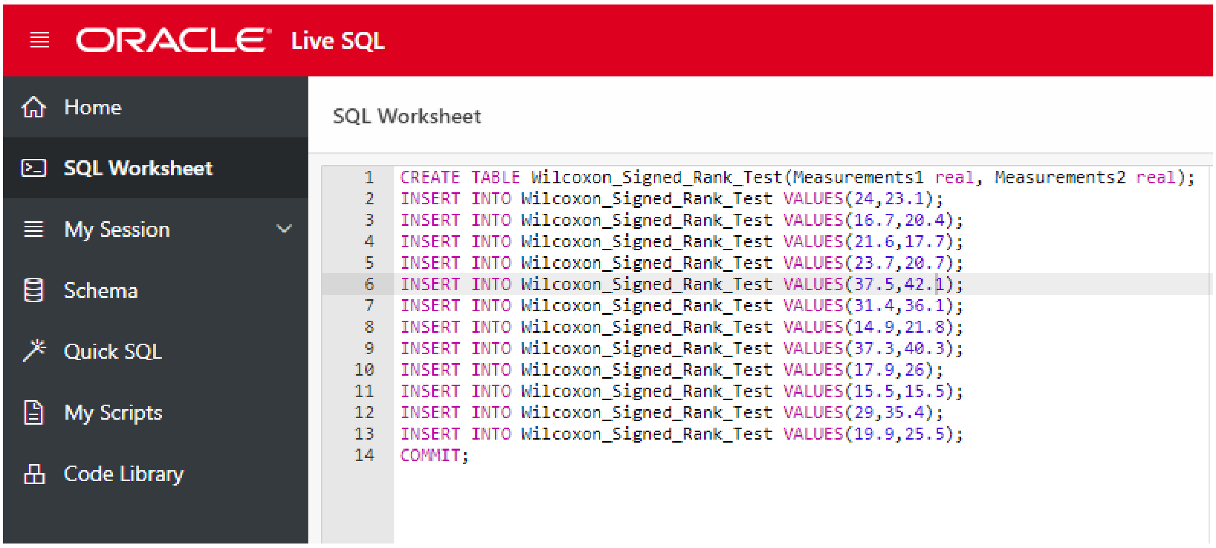


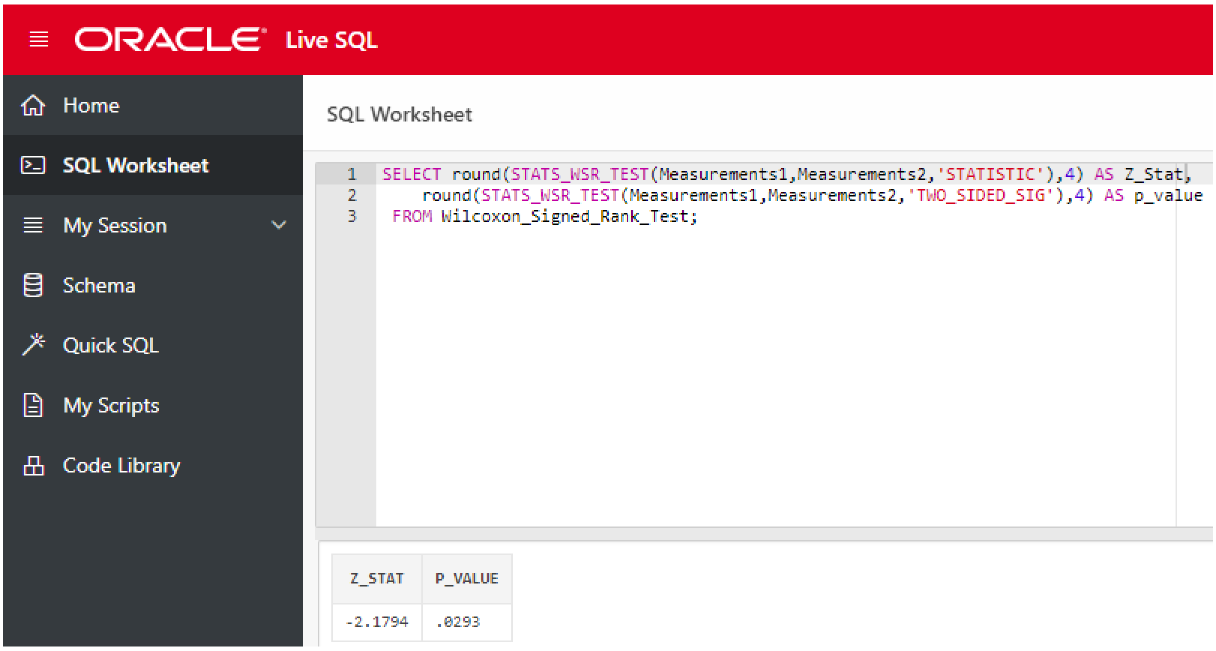


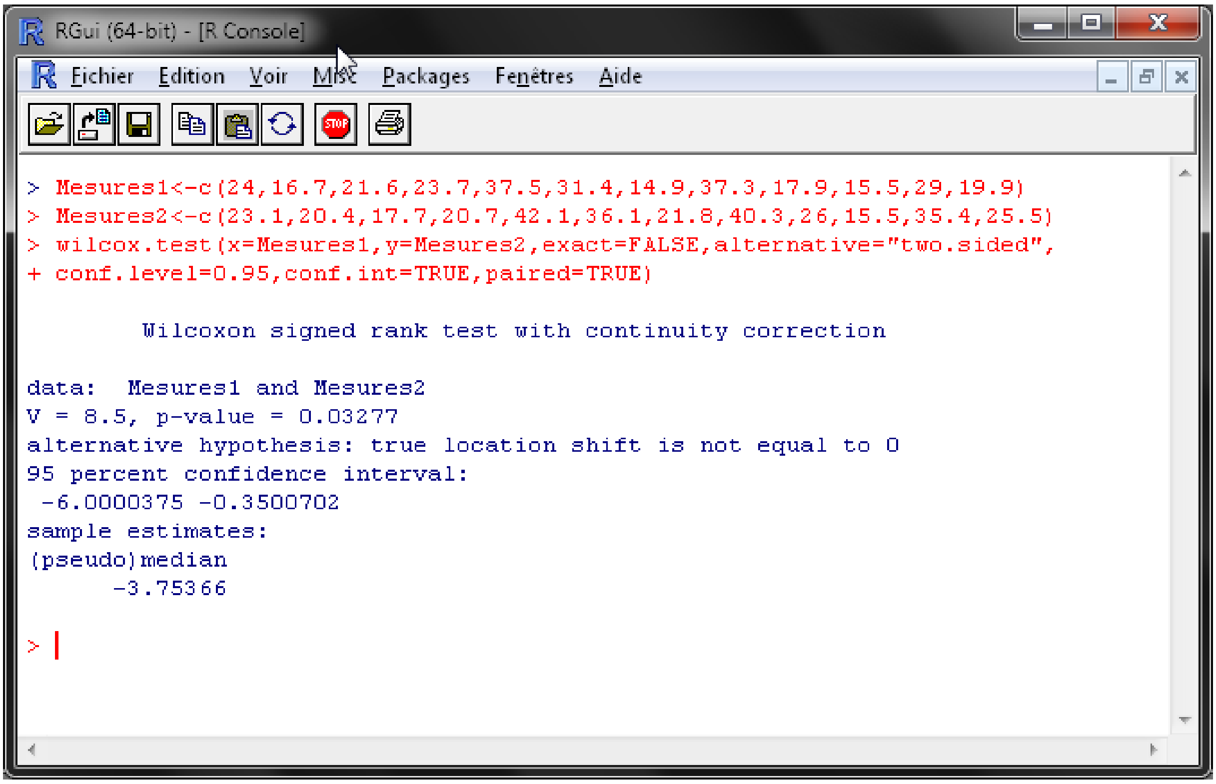

RGui (64-bit) - [R Console]
Fichier Edition Voir Misc Packages Fenêtres Aide
> setwd("C:/")
> mydata<-read.csv("Pipelines.csv",header=T,sep=";")
> t.test(mydata$Pipeline1,mydata$Pipeline2,var.equal=F)
Welch Two Sample t-test
data: mydata$Pipeline1 and mydata$Pipeline2
t = 1.2398, df = 34.797, p-value = 0.2233
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-3.423986 14.160828
sample estimates:
mean of x mean of y
160.3684 155.0000
>

11.12 Wilcoxon signed rank test

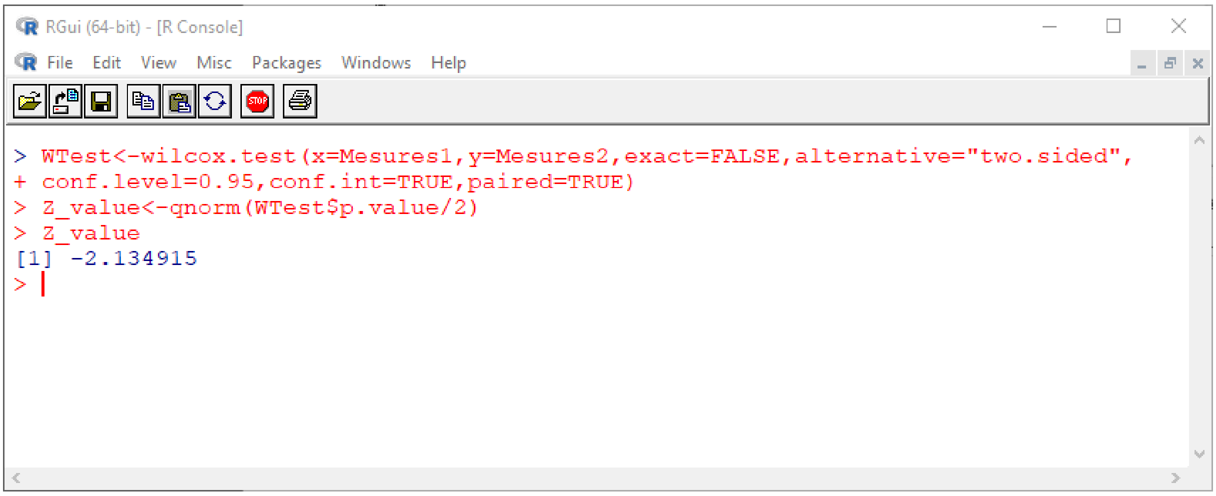








On peut obtenir la valeur de Z proche de celle que renvoie Oracle avec la commande suivante:



12 List of Figures

13 List of Tables

14 Index